The Computational Geometry of Hydrology Data in Geographic Information Systems

by

Michael McAllister

B. Math. (Computer Science, C & O), University of Waterloo

M. Sc. (Computer Science), University of British Columbia

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

we accept this thesis as conforming
to the required standard

The University of British Columbia

December 1999

© Michael McAllister, 1999

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science
The University of British Columbia
Vancouver, Canada

April 24, 2000

Date

Abstract

Computational geometry counts geographic information systems (GIS) among its application areas. GIS data sets are large and are related to the Earth's surface by geometric coordinates. These properties mesh with geometry algorithms and their asymptotic analyses. This dissertation investigates the geometry in a specialized set of GIS problems, namely finding river network centrelines and finding watershed boundaries. Our goal is to create robust and consistent algorithms that solve these problems efficiently.

When finding river network centrelines, the main issue is the robustness of the algorithm. The medial axis is an excellent centreline for a river or lake, but few robust implementations exist. Moreover, the medial axis uses parabolic segments, which are harder to represent accurately than line segments. We approximate the medial axis with a piecewise-linear structure that we compute with a robust algorithm for point Voronoi diagrams. Our approximation provides estimates of river area and associates the points on the centreline with the nearest river bank points, much like the medial axis. In turn, we use this association to identify opposite points along river banks.

When finding watershed boundaries on triangulated terrains, we focus on finding an algorithm that is consistent with the terrain. In previous work, Nelson et al. [66] locate regions that drain to a common location, but their solution can be disconnected in degenerate cases. We propose a vector algorithm whose watersheds are consistent with the assumption on how water flows on the terrain; the algorithm guarantees one polygon per watershed. Our algorithm finds the watershed boundaries for local minima in the terrain in $O(n \log n + k)$ worst-case time where n is the number of points that model the terrain and k is the complexity of the watershed boundaries.

We apply our solutions for these GIS problems to data in the British Columbia TRIM (Terrain Resource Inventory Mapping) format. For river networks, we find centrelines for the

Fraser River in British Columbia and for a subset of its tributaries. We link these centrelines into a single river network. We have used this network to model the migration of salmon. For watersheds, we identify the boundaries of watersheds in the mountains north of Vancouver, British Columbia and compare the boundaries with manually-digitized watersheds of the same region.

Contents

List of Figures

Acknowledgements

I owe many thanks to my supervisor, Jack Snoeyink. Throughout my years in graduate school, he played many roles for me. He was an adamant teacher who pushed me to bring out the best of my work. He taught me to look for interesting problems no matter where I was—there is always one more to find. He had the time, patience, and knowledge to answer endless questions when I was stuck. He was also a friend and gave me numerous pep talks when I was discouraged. I could always rely on him and he never gave up on me. Thanks Jack. I also want to thank David Kirkpatrick. Besides being on my supervisory committee and giving me a lot of feedback on my dissertation, David was always a calming presence for me. He listened to my problems, both technical and personal, and had excellent advice for every occasion. With Alain Fournier and Brian Klinkenberg completing my supervisory committee, I was confident that I could solve just about any problem that came my way. At my defense, the other examiners, consisting of Jim Little, Stephen Sheppard, Robert Miller, and Nicholas Chrisman, raised many interesting ways to extend and improve my dissertation. I thank them all for their thoroughness.

My dissertation work balances theory and practice. The theory group in computer science at UBC provided a mix of talks and problem sessions that expanded the limits my knowledge. I was fortunate to have friends with whom I could discuss theoretical problems: Jit Bose and Will Evans in the early years of my degree and Alex Brodsky, Bettina Speckmann, and Stephane Durocher in the later years. Meanwhile, Ian Williams and the all people at Facet Decision Systems kept my feet on the ground with their application of river centreline networks to modeling salmon migration in the Fraser River. Ian's patience and perseverance through late night debugging sessions of river networks and through many revisions of my software kept the project alive. Dave Hawkins believed in fostering a relationship between industry and academia and spent too many weekends doing the high-level modeling on top of my river networks. Finally, Gerry Furseth always had time to answer

my technical questions, no matter how chaotic his day was.

Everybody gathers a core group of friends who support them throughout their degree. Aside from Brian Edmonds' endless technical help, he kept me well-fed with books to read in my spare time. I could always count on Scott Flinn to accompany me on a day-long bicycle ride or to challenge me to a game of squash. Karen Flinn kept our group together by organising Tuesday night movies. Nancy Day entered the PhD program at the same time as I did, was a constant companion in the office down the hall, and showed me that it really was possible to finish my degree. There was never a dull moment whenever Jennifer Shore was around; I could count on her to throw around a disc, a baseball, or a football, or just to sit around and talk. Ian Cavers wraps up my group of close friends. He would listen to my problems, especially when I was teaching, and offer excellent suggestions.

Moving from Ontario to British Columbia was difficult for me. My cousins Christine, Tim, Brad, and Erin Corkan gave me a home away from home in Vancouver. Although I hadn't seen them in many years, they welcomed me into their home. When I got homesick, I could always drop in for a visit and they would cheer me up. They were a lifeline that I depended on.

Most of all, I thank my brothers and sisters, Shawn, Dayna, Dean, Dennis, Jennifer, and Sheryl, and my parents, James and Donna, for their continual encouragement and faith. I cannot describe all the ways in which they have helped me. They picked me up when I felt down. They put setbacks into perspective for me and shared my joy when things went well. They believed in me.

This thesis was completed with financial support from the Natural Sciences and Engineering Research Council of Canada (NSERC), the BC Advanced Systems Institute, a UBC Killam scholarship, the IRIS National Centre for Excellence, and Facet Decisions Systems of Vancouver.

MICHAEL MCALLISTER

The University of British Columbia
December 1999

Chapter 1

Introduction

The challenge of computational geometry is to exploit the geometric structure in a problem and obtain an efficient solution, whether as an algorithm or as a data structure, or prove that the problem is "hard" in some respect. Computational geometry problems can be motivated by either fundamental issues, such as whether or not there is a linear time algorithm to triangulate a polygon [15] or by practical issues, such as point-location data structures [8, 16, 32, 55], pattern layout algorithms [62], and industrial part casting algorithms [13, 77]. The work in this dissertation is motivated by a hydrology problem from geographic information systems.

A geographic information system (GIS) is a system for

> "capturing, storing, checking, integrating, manipulating, analyzing and displaying data related to positions on the Earth's surface." [4]

Computational geometry algorithms count GIS problems among its application areas since the characteristics of the problems match the goals in computational geometry:

- algorithmic efficiency is important,
- the quantity of data justifies asymptotic analyses of algorithms, and
- the problems have a geometric aspect.

The geometry in GIS problems occurs under in many forms that include explicit adjacencies or intersections of lines, implicit regions between lines, and interpolations between known data sources. Computational geometers define and solve different problems that have similar geometric

characteristics: polygon union and intersection, map labeling, multi-scale terrain representation, interpolation schemes, proximity graphs, and embedded graphs of lines, to name a few. Solutions to many of these geometric problems become tools for GIS users.

The problem addressed by this dissertation originated with the Geographic Data BC division of the BC Ministry of Environment, Lands, and Parks. This division controls the government's paper and digital maps for British Columbia. One of their classes of maps is the BC watershed atlas that contains a digital copy of the watershed boundaries derived from 1:50 000 scale maps. These watersheds are fundamental in managing natural resources in the province. In the early to mid 1990's, the BC government began to digitize 1:20 000 scale maps of the province. This new data standard would provide finer resolution digital data than the existing 1:50 000 scale maps. The problem was in migrating current databases to fit the new maps. For information that was derived from the 1:50 000 scale maps, such as the watershed atlas, the government could overlay it directly on the 1:20 000 scale maps, could re-digitize it from the new maps, or could compute it from the electronic data. Applying the existing watershed boundaries to the 1:20 000 scale maps creates inconsistencies with features that were not detectable on the 1:50 000 scale maps. Re-digitizing the watersheds would be a tedious undertaking. Consequently, we examined the problem of identifying the watershed boundaries directly from the 1:20 000 scale map data.

When identifying watershed boundaries, we focus on two issues: *robustness* and *consistency*. First, we say that an algorithm is *robust* if it is able to produce meaningful results in the presence of degeneracies. In computational geometry, a degeneracy is a property of the data such as three collinear points or four cocircular points. For simplicity, computational geometry algorithms often assume that their data is in general position; we assume that degeneracies do not occur in the data. This assumption does not apply to GIS data. Second, we say that an algorithm is *consistent* if it is faithful to its underlying assumptions. If these assumptions do not cover all possible cases such as degeneracies or the model under the assumption is wrong such as using an approximation to a terrain rather than the terrain itself, then the results of the algorithm may seem incorrect yet still be consistent. In these latter cases, the fault for the error does not lie with the algorithm itself.

The vector data in a BC TRIM 1:20 000 scale digital map contains two sources of information for computation of hydrological features: planimetric data and elevation points. The planimetric data is a set of polygonal line tracings of river banks and lake shores. The elevation points are a

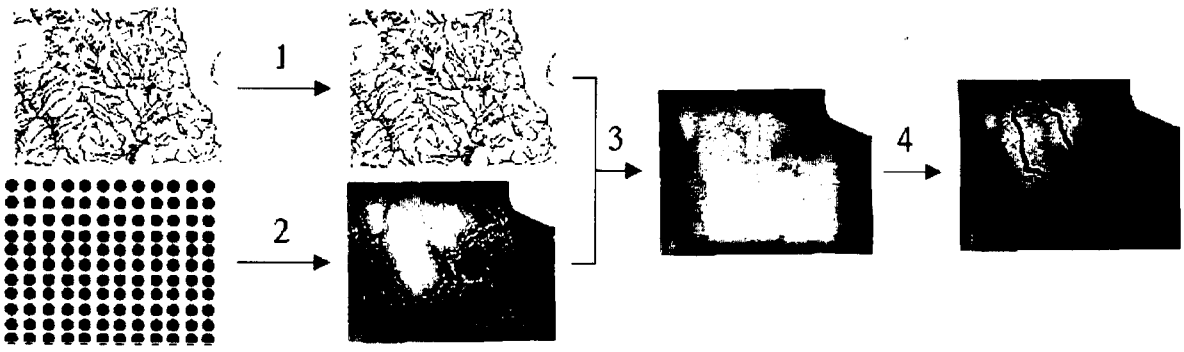


Figure 1.1: Data workflow: 1. derive centrelines, 2. derive a TIN, 3. embed rivers as TIN breaklines, and 4. identify watersheds

set of irregularly spaced 3D points that let us model the terrain. These sources of data lead to a natural breakdown of the watershed problem into two components (Figure 1.1 arcs 1 and 4): simplify the river networks into a centreline network and combine this network with the terrain elevations to identify the watershed boundaries.

For the first component, we develop a robust algorithm that approximates the medial axis of rivers and lakes. We model explicit river banks as a graph and capture the implicit river with our approximation. We motivate the approximation from an analysis of river networks and their geometric properties. As a side-benefit of our river approximation, we defined a model for gauging the effects of water temperature and current strength on migrating salmon in the Fraser River of British Columbia. This model was done in conjunction with the Department of Fisheries and Oceans in Nanaimo. We also derived river characteristics from the geometric foundation of our approximation.

For the second component, we define a data structure, called a *watershed graph*, that abstracts watersheds in terrains and prove that the data structure is consistent with the terrain model. We use a triangulated irregular network (TIN) for the terrain and model water flow as paths of steepest descent on the TIN. Current vector algorithms for detecting watersheds empirically compute valid watershed boundaries but violate the water flow model in degenerate data instances. We prove that our watershed graph correctly represents the watersheds of a TIN, even in the presence of certain degeneracies in the TIN. We present an optimal, straight-forward algorithm to create the watershed graph. Along with the algorithmic solutions to the centreline and watershed problems, we report on practical implementations of the algorithms at the end of the relevant chapters.

Our algorithm for watershed graphs is derived solely from the TIN. We assume that the river centrelines are already embedded in the TIN as breaklines to adjust the drainage properties. To gauge the effectiveness of these breaklines in guiding our watershed algorithm, we report all but one example from TINs that do not contain the rivers.

In Chapter 2 we review graphs and data structures from mathematics and computer science that underlie our solutions and also review some basic river terminology. In Chapter 3 we present our river centrelines as approximations to the medial axis. We review terrain representations, particularly the triangulated irregular network (TIN), and their role in identifying watersheds in Chapter 4. In Chapter 5 we define the watershed graph for a terrain and prove consistency properties about the graph. Finally, we summarise our results and discuss future directions in Chapter 6.

Chapter 2

Geometric and River Networks

There are two sub-tasks in our study of river networks. First, we must store and display the river networks efficiently. For the network storage structure, we want find the interconnections between the network edges. For the display, we want to trace features, such as lakes, so that they can be coloured on maps easily. Both the storage and the display are determined by the representation used for the river network. Second, we must analyse the river network. More specifically, we want to extract implicit information from the network, such as the interconnections between rivers and opposing river banks. This information relates to the *topology* of the river network; the physical paths that a river follows, called the *embedding* of the river, is just one part of the topology. From the topology, we can then derive metric information such as the area of rivers.

This chapter reviews background concepts from computer science and from hydrology that deal with the topology and embedding of rivers. In Sections 2.1 and 2.2 we describe the underlying graphs and data structures represent rivers. In Section 2.3 we present some of the differences in how computational geometry algorithms and a GIS operations treat their data. Section 2.4 reviews methods for simplifying data; for rivers, this applies to the embedded edges of the river network and does not change the topology. Then, in Sections 2.5 and 2.6, we discuss river networks orders. Sections 2.1, 2.2, and 2.3 will be familiar to computer scientists while Sections 2.4, 2.5, and 2.6 will be more familiar to GIS practitioners.

2.1 Graphs

A *graph* represents a relation between two or more objects or concepts. The objects are *vertices* of the graph (also called *nodes*) and the presence of a relation between two objects appears as an *edge* between the two vertices for the objects. More formally, a graph G is defined by a vertex set V and an edge set $E \subseteq V \times V$. An edge (a, b) represents a relation from vertex a to vertex b. This edge definition implies that every edge has a direction, so edge (a, b) is distinct from edge (b, a). These graphs are called *directed graphs*. For example, the relationship between people where a is related to b if a has borrowed something from b defines a directed graph.

Many relations are symmetric: if (a, b) is an edge then (b, a) is also an edge. While a directed graph can represent a symmetric relation, it contains redundant edges since every edge (a, b) has a matching edge (b, a). In this case, we omit the distinction between the two edges and represent both edges by a single pair (a, b). We call this graph an *undirected graph.*

Graphs are abstract; they do not contain any information on how to draw them. A placement of vertices and edges in $\mathbb{R}^2$ for a graph is called an *embedding* of the graph. Figure 2.1 shows several different embeddings in $\mathbb{R}^2$ for the same graph where every edge is the straight line between its vertices. Once a graph is embedded, we define the *faces* of the graph as the regions of the plane enclosed in graph edges.

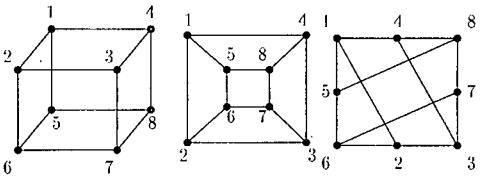
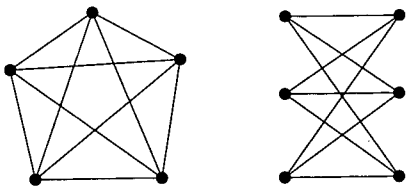


Figure 2.1: Different embeddings of one graph in $\mathbb{R}^2$

We can trace a face of the graph by starting at an edge, following that edge to one of its endpoints, following the next counterclockwise edge around the vertex, and repeating this process until we return to the starting edge. The middle graph of Figure 2.1 has six faces: vertex sequences 1265, 2376, 3487, 4158, 5678, and the outer face 1234. In Section 2.5 we model river networks with planar directed graphs where lakes and wide rivers are faces of the graph. We also attach attributes to the graph edges, such as river names, and use the edge directions to indicate the direction of water flow along the edge.

When a graph can be drawn or embedded in the plane without having edges intersect except at common endpoints, whether the edges are straight lines or curves, then the embedding is a *planar* embedding and the abstract graph is planar. Not all graphs are planar; the smallest non-planar examples are the complete graphs K_5 and $K_{3,3}$. (A complete graph is a graph with an edge between every pair of vertices). The middle embedding in Figure 2.1 is a planar embedding. Even though the other two embeddings are not planar, the graph is planar since it has at least one planar embedding. The planarity condition restricts the number of edges or faces of a graph according to Euler's formula: $f - e + v - 1 = 1$ where f is the number of faces, e is the number of edges, and v is the number of vertices for the graph. As a consequence, a planar graph with n vertices has at most $3n - 6$ edges and $2n - 4$ faces. This maximum number of edges and faces occurs in a *triangulation* where every face of the graph is a triangle (Figure 2.3a).



K_5 $K_{3,3}$

Figure 2.2: Complete graphs K_5 and $K_{3,3}$

Trees are a family of graphs in which there is a unique path (that uses each edge of the graph at most once) between every pair of vertices (Figure 2.3b). The characteristics of trees can simplify graph algorithms such as the network orders of Section 2.6. A *leaf* of a tree is a vertex with only one incident edge.

(a) (b) (c)

Figure 2.3: A triangulation (a), a tree (b), and an acyclic directed graph (c)

Another family of graphs is *acyclic graphs* (Figure 2.3c). An acyclic graph is a directed graph that does not contain any oriented cycle. For example, since a path between pairs of vertices in a tree is unique, any orientation on the edges of a tree is a trivial example of an acyclic graph.

Any acyclic graph defines a partial order $\prec_p$ on its vertices. For graph vertices a and b, the relation $a \prec_p b$ holds if and only if directed edge (a, b) is in the graph. We say that the vertices a and b are comparable. Not all vertices in a partial order are necessarily comparable. A topological order (or total order) $\prec_t$ for the same graph makes every pair of vertices in the graph comparable (whether or not there is an edge between the vertices in the graph), maintains transitivity, and

respects the partial order—if $a \prec_p b$ then $a \prec_t b$. Topological orders are useful for propagating results through a graph; they ensure that the vertices at the incoming edges to a vertex v are processed before vertex v is processed. We use topological orders for this purpose in Section 2.6.

2.2 Graph Data Structures

Different representations for graphs balance storage space against the time needed to find whether or not a particular edge exists. There are several ways of storing a graph. The simplest way stores the edges of the graph as a list of ordered vertex pairs. A more complex way stores the edges with an *incidence matrix* where the matrix has one row for each vertex, one column for each edge, and entry (i, j) of the matrix is 1 if and only if edge j has vertex i as an endpoint. An *adjacency matrix* is a more space-efficient representation than the incidence matrix when there are many edges. The adjacency matrix has one row and one column for each vertex. Entry (i, j) of the matrix is 1 if and only if vertex i and vertex j are the two endpoints of an edge in the graph. An *adjacency list* improves on the space requirements for storing graphs with few edges: each vertex of the graph has a list of its adjacent vertices.

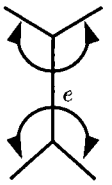


Figure 2.4: Winged-edge structure pointers

Once a graph is embedded, the order in which edges enter a vertex determines the faces of the graph. The *winged-edge* [7] or *quad-edge* [46] data structures capture this information. Both structures are edge-based, meaning that they explicitly store the edges of the graph, and use comparable amounts of storage space.

The winged-edge data structure is similar to a doubly-linked list. Each edge keeps a reference to the next clockwise edge and the next counterclockwise edge around each of its endpoints (Figure 2.4). As with linked lists, two of the four references for each edge are redundant. For example, you can find the next clockwise edge around an endpoint by following the counterclockwise edge pointers until you were about to return to the original edge. The redundancy allows for constant-time access to the edges immediately preceding or following an edge in any face of the graph embedding.

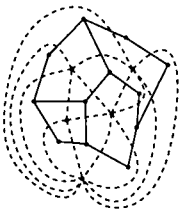


Figure 2.5: A graph (solid) and its dual (dashed)

Although an edge in a planar graph usually represents a relation between its end vertices, we can interpret the edge as a relation between the faces on either side of the edge instead. Let G

be an embedded planar graph with vertices V, edges E, and faces F. We define the graph G' from G as the graph with vertices F and edges E' where $E' \subseteq F \times F$ and (u, v) is an edge of E' if and only if there exists an edge $e \in E$ that bounds face u and v in G. There is a bijection between the edge sets E and E' (Figure 2.5). The graph G is the *primal* graph and G' is its *dual* graph [9].

The quad-edge data structure simultaneously represents the primal and dual aspects (vertices and faces) of the graph in a symmetric manner [46]. We expand on duality in Section 2.7 when we describe Voronoi diagrams and Delaunay triangulations. As with the winged-edge data structure, each edge in the quad-edge data structure keeps a reference to the next counterclockwise edge around its endpoints. For each edge e, there is a dual edge e' that uses the faces to either side of e as its endpoints. Edge e also keeps references to the next counterclockwise dual edges around each end of e'. This scheme also allows for constant-time access to the edges immediately preceding or following an edge in either the primal or dual graphs. For example, to find the edge clockwise of edge e in Figure 2.6, take the next counterclockwise edge d' from the dual edge e'. The next clockwise edge of e is the edge d that has d' as its dual edge.

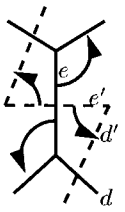


Figure 2.6: Quad-edge structure pointers

2.3 Geometric and Geographic Data

Computational geometry algorithms and geographic information systems (GISs) both transform raw data into more useful data structures. In many instances, a geometric transformation is only a part of a whole GIS operation. Both disciplines assume that the input to their transformations does not contain any ambiguities, but they differ in what they consider ambiguous.

Ambiguities for computational geometry algorithms involve degeneracies in the geometry of the data. Examples of these degeneracies are 2 distinct points with identical geometric coordinates, 3 collinear points, or 4 cocircular points. These degeneracies affect the topological structure of the data. Most computational geometry algorithms assume that degeneracies do not occur for ease of algorithm presentation and analysis.

Ambiguities for GIS operations involve duplications of complete data features and conflicts between attributes of features. By the nature of GIS data, some geometric degeneracies such as collinear points are unavoidable. Consequently, GIS operations cannot completely ignore the geo-

metric degeneracies in their inputs. To use computational geometry algorithms on the geometric component of their data, GIS operations transform their data in phases. First, they ensure logical consistency within the data, whether by resolving feature duplication, semantic contradictions between features, geometric conflicts or attribute mismatches. Second, they apply geometric transformations to describe the data. Third, they transform attributes from the original data to describe the derived features. Chrisman [22] presents further information on classifying GIS operations as data transformations; Chrisman's paper describes a method for viewing GIS operations as having geometric, temporal, and attribute components that must be balanced in order for a GIS operation to be meaningful.

Logically consistent data is often assumed in computational geometry algorithms. We can view the step of obtaining logical consistency as distilling quality data from a quantity of data. We present one example of this process, data simplification, in Section 2.4. We define the constraints for logical consistency of river networks in Section 3.2 prior to the actual computational geometry algorithm for river centrelines in the rest of Section 3.

2.4 Data Simplification

Since we collect and display map data at different scales, we must adapt the content of the map for clarity. This section gives pointers to general work on this adaptation process and then focuses on one specific aspect, the simplification of polylines, in Section 2.4.1.

Map generalization is the "simplification of map information so that information remains clear and uncluttered when map scale is reduced" [4]. Map generalization can involve many operations. Weibel and Brassel [14] and McMaster and Shea [60] classify the basic operations according to why, when, and how to generalize. Weibel([89], p. 126) lists a sequence of basic operations for cartographic generalization:

- *Selection/elimination:* Is applied first as it eliminates insignificant details and features and increases available space.
- *Aggregation/amalgamation/merging:* These operators combine selected features and thus save space. Additionally, they induce a transition of topological type (e.g., point to area, double lines to single line, area to line) and thus must precede line processing operators (simplification, smoothing, etc.).
- *Simplification:* Reduces detail and contributes to line caricature. Should therefore be applied at an early stage.

- *Smoothing:* Contributes to aesthetical refinement. Follows simplification.
- *Displacement:* Used to resolve spatial conflicts created by previous operators.

Chapter 3 focuses on one operation, collapsing rivers, for generalization and network analysis. The work assumes that data simplification has already been performed; the rest of this section presents the background operations for the simplification of polylines. For further details on general simplification, see Weibel's presentation on generalization at the workshop on the algorithmic foundations of GIS [92].

High-level details on polylines in small scale maps are not necessary since they cannot be seen. Detailed data also affects the performance of algorithms since the speed of algorithms is a function of its input size. Data simplification reduces the detail for small scale maps and reduces the input size for algorithms. For maps, simplified data uses fewer points for each polyline without changing the visual quality of the feature. For algorithms, data simplification reduces the input size. When polylines represent features such as rivers, we cannot discard individual lines to decrease the input size or to improve the map clarity. Instead, we approximate each polyline (cartographic *simplification*) by reducing the number of points without changing the characteristics of the polyline too drastically. For example, the overall curvature of a river should not change with the simplification.

For simplification of polylines, we decrease the accuracy of the data without changing its precision. Accuracy refers to "the closeness of observations, computations or estimates to the true value as accepted as being true" [4]. Precision refers to "the exactness with which a value is expressed, whether the value be right or wrong" [4]. For example, a location with coordinates specified to 3 decimal places is more precise than the same location with coordinates specified to 2 decimal places but is not necessarily more accurate. In simplification, we keep a subset of the features or a subset of each feature, which decreases accuracy. With a subset of features, we omit complete polylines when they are deemed unnecessary. Although we can do this for displaying river networks (Section 2.6), river analyses involve the smaller streams and these streams must remain in the calculations. One example here is our model for fish migration. With a subset of each feature, we omit points within a polyline; at worst, the polyline is left with two points. The rest of this section focuses on algorithms that select a subset of points from a polyline as its simplification.

2.4.1 Polyline Simplification

The goal of polyline simplification is to select a subset of the points from a polyline as a coarser representation of the polyline. We call the coarser representation a *simplified polyline*. We restrict ourselves to selecting vertices of the original polyline for the simplified polyline; we cannot select a point along an edge of the polyline as a vertex of the simplified polyline.

When simplifying a polyline, we want to preserve the cusps and curvature of the polyline while using as few points as possible. The user controls the trade-off between descriptive detail in the simplified polyline and the number of points with a simplification parameter called the *simplification tolerance*. Different algorithms map the simplification tolerance to a *neighbourhood* for each point; the neighbourhood of the i th point of the polyline is the sequence of points $i-j, \ldots, i, \ldots, i+k$ for positive integers j and k. As we shall see, some algorithms define a fixed-size neighbourhood for a point where $j = k$ while other algorithms require that the simplified polyline remain within a set distance from the original polyline.

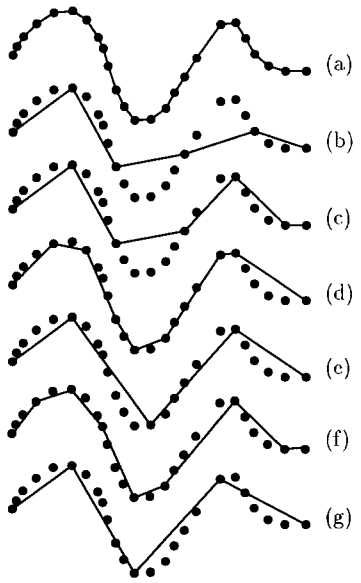


Figure 2.7: A line and its simplifications: original line (a), n th point (b), minimum distance (c), minimum area (d), fixed look-ahead (e), fixed-orientation corridor (f), and Douglas Poiker (g)

The earliest polyline simplification methods did not preserve polyline characteristics. The simplest algorithm selects every nth point as a simplified polyline, which treats every point of the polyline independently of all others (Figure 2.7b). McMaster [61] proposed a point-decimation algorithm with a slightly wider neighbourhood for a point (Figure 2.7c). A local criterion, such as the distance to the nearest adjacent neighbour or the angle of the polyline at a vertex, characterises the importance of each point on the original polyline. The algorithm removes the points of low importance to simplify the polyline. With McMaster, the elimination is a greedy decimation in a single pass of the polyline and leads to a linear-time algorithm. The algorithm could equally-well iteratively remove the least important point until the approximation satisfies an error-tolerance criterion. Although these two algorithms are fast, they do guarantee that the simplified polyline preserves any polyline characteristics.

Visvalingam and Whyatt [91] propose a different decimation algorithm. Their algorithm

uses an approach similar to the McMaster variation where the least-important point is removed from the polyline. Their importance criterion is the effective area of a point—the area formed by the triangle of a point and its two neighbours (Figure 2.7d).

Lang [54] extends the notion of a neighbourhood of a vertex v to a fixed number of vertices away from v along the polyline (Figure 2.7e). Lang's algorithm with a neighbourhood size d evaluates how well the line segment from the i th vertex to the $i + d$ th vertex approximates the polyline between the vertices. Given a user-specified tolerance ϵ, the approximation is good only if all the vertices between the i th and the $i + d$ th are within ϵ of the approximating line segment. When the approximation is bad, the algorithm shrinks the neighbourhood (from d to $d - 1$) until the approximation is good. The algorithm bounds the distance between the original polyline and its simplification by a sequence of ϵ-corridors around the polyline (rectangles of height 2ϵ).

Subsequent algorithms maintained Lang's ϵ corridor as a bound on the error and removed the size limit of a vertex neighbourhood. From vertex i of a polyline, Reumann and Whyatt [91] create an ϵ corridor parallel to the segment between vertices i and $i + 1$, find the first edge of the polyline, say between vertices j and $j + 1$, that crosses the boundary of the ϵ corridor, and approximate the polyline from vertex i to j with the straight-line segment between vertices i and j (Figure 2.7f). A vertex-stabbing algorithm by Guibas et al. [44] removes the restriction that the corridor be parallel to the edge between vertices i and $i + 1$.

Alternatively, we can consider each point relative to the entire polyline. Douglas and Peucker (now Poiker) developed a simplification algorithm with a global neighbourhood [29, 48]. Given an error tolerance ϵ and a polyline l, their algorithm generates a polyline l' that uses a subset of the points of l. Every point of l is within ϵ of its nearest edge on l' (Figure 2.7g and Figure 2.8). The notion of the ϵ-tolerance traces back to a paper by Perkal [72]. Unlike the algorithms of Lang and of Reumann and Whyatt, Douglas and Peucker's algorithm adds points to an approximation instead of removing them from the original polyline. The algorithm approximates the polyline with the line segment between the first and last points. The vertex of the original polyline that is farthest away from the approximation is added to the approximation. The process

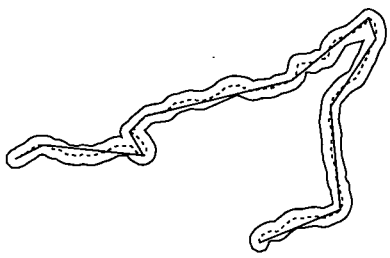


Figure 2.8: Douglas-Peucker simplification (solid) of a line (dashed) and an ϵ-tolerance around the original line.

of adding the farthest vertices continues until all vertices of the original polyline are within ϵ of the approximation. The Douglas Peucker algorithm keeps the important inflections of the polygonal lines while decreasing the number of points within the polygonal lines.

These simplification algorithms balance optimization with speed of computation. The algorithm that selects every n th point is the fastest and easiest algorithm but performs no optimizations. At the other end of the spectrum, Cromley and Campbell [24, 25] extend the polyline into a complete graph, direct the graph edges in the same order as the endpoints appear in the polyline, weigh the edges according to an optimization criterion (such as the area between the approximation line and the original polyline), and use optimization algorithms to achieve a minimum cost approximation. Such an algorithm could use the entire graph, whose size is quadratic in the length of the polyline, and pays the consequent cost in time complexity.

Each of these simplifications handle polygonal lines in isolation. They do not prevent separate lines from intersecting one another after the simplification. This form of collision detection and avoidance is often handled as a separate part of map generalization.

2.5 River Topology

Planar directed graphs are convenient representations for river networks. We define the graph by its vertices and edges; we call the drawn or embedded river lines *arcs* to distinguish them from the abstract graph edges. The vertices of the graph are river junctions and a directed edge from junction a to junction b represents a river that flows from a to b. If we place the graph vertices at the geographic locations of the river junctions and force the edges to follow the river arcs between the junctions then we have a planar embedding for the graph. With more detailed data, we can represent lake shores and riverbanks as separate graph edges, which make lakes and wide rivers into graph faces.

In basic network representations, we use one graph edge for each individual line segment in a river arc. Consequently, the junction nodes in a river network only have two incident edges where a river simply continues to flow. To reduce the number of graph edges, other work [50] approximates long river arcs by the straight-line arc between two river junctions or between a source and the first downstream river junction (Figure 2.9).

In our work, we separate the topological connections of river edges from their geometric

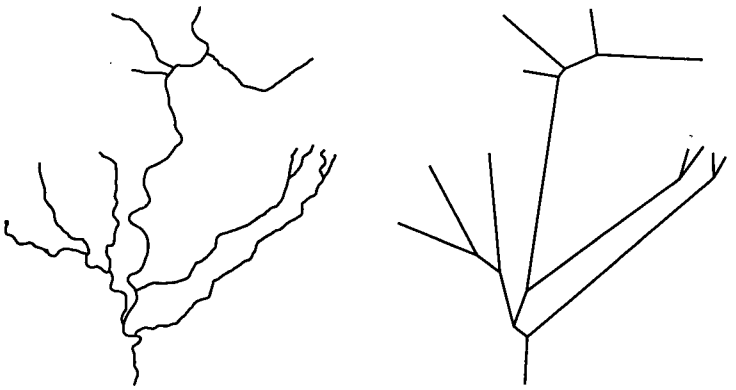


Figure 2.9: A path embedding and a straight-line embedding of a river network

expression on a terrain. We store the arcs between river junctions as a single graph edge in a quad-edge data structure [46] and link a polygonal line to each edge as its geometric expression. Consequently, one quad edge can have a sinuous path without expanding the size of the network topology graph.

2.6 River Network Orders

Some river network analysis treat the rivers as trees rooted at their downstream point [49, 80, 83]. These analyses use the amount of branching upstream from the origin vertex of an edge as a measure of the edge's importance in the tree. The ordering of network edges by branching factor defines a *network order* as an attribute for each river edge. We restrict our attention to three network orders as depicted in Figure 2.10: Strahler, Horton, and Shreve. The Horton order finds primary river channels, called the river mainstems, and prioritizes the mainstems by their branchings. As we shall see, the Horton order is strongly related to the Strahler order. Once computed we can view the order of an edge as just another attribute of the edge in the graph.

We use a topological order of the edges to compute each network order. The proliferation of the "order" terminology is an unfortunate consequence of drawing on standard concepts from two disciplines. While we interpret the network orders as attributes of river edges, a topological order of a network is a sequencing of the network edges. For the topological order, we treat the river network as an acyclic directed graph whose edges are oriented from upstream to downstream and use the topological order of the graph as defined at the end of Section 2.1.

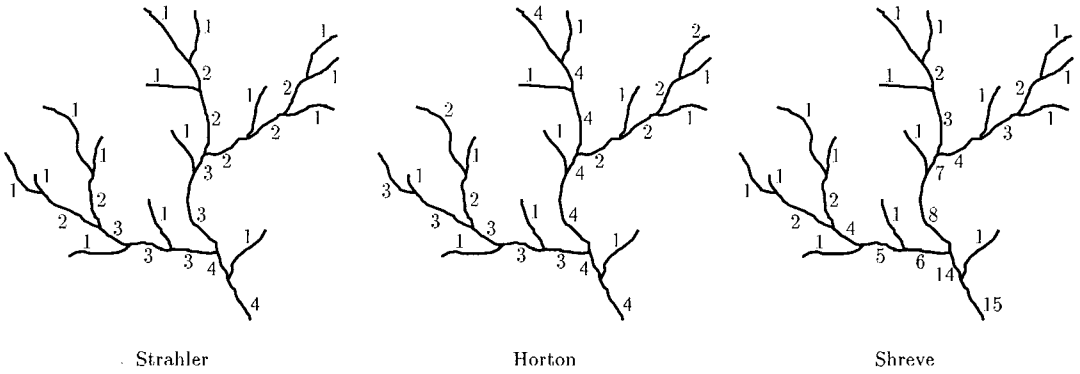


Figure 2.10: Strahler, Horton, and Shreve network orders

The Strahler order [83] derives its network order from the topology of the river network:

- The Strahler order of a river edge with no upstream rivers is 1.
- The Strahler order of a river edge e with upstream rivers depends on the orders of the upstream rivers. Let m be the largest order for the upstream rivers. If there is only one river at the upstream end of e with order m then the order of e is m. Otherwise, the order of e is $m + 1$.

As the definition suggests, a single-pass of the river network that follows the topological order of the edges will compute the Strahler order.

The Horton order [49] is similar to the Strahler order. Conceptually, the Horton order propagates Strahler order values upstream to identify a river *reach*, namely the longest stretch of open water that forms a river upstream from a point:

- The Horton order of a river edge with no downstream rivers is the same as the Strahler order for the river edge.
- The Horton order of a river edge e depends on the longest river that enters its downstream vertex. If e does not belong to the longest path upstream from downstream vertex of e then the Horton order of e is the Strahler order of e. Otherwise, the Horton order of e is the Horton order of the unique river edge that leaves the downstream vertex of e.

This definition suggests a two-pass algorithm to compute the network order. The first pass follows the topological order of the network to obtain the Strahler order and to accumulate the upstream river lengths for each edge. The second pass reverses the topological order of the network to propagate the largest Strahler order values upstream in the network.

The Horton order can be computed in a single pass of the river network. In addition to computing upstream river lengths with the Strahler orders, we label each edge with a reach identifier. A river edge with no upstream river edges has a unique reach identifier. A river edge with upstream river edges gets the reach identifier of the longest river that enters its upstream point. At the same time, we calculate the Strahler order values on the reaches instead of on the edges.

The Shreve order [80] for a river edge e is the number of leaves that are upstream from e. As with the Strahler order, a single pass through the topological order of the network order will compute the Shreve order.

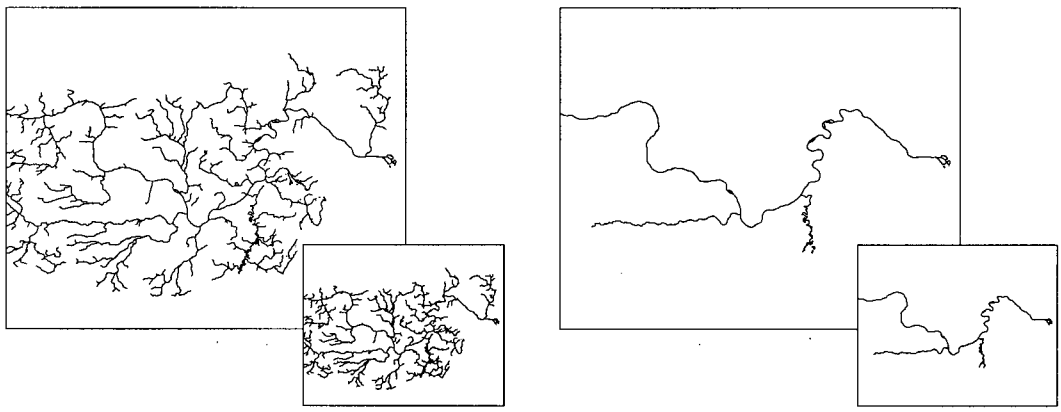


Figure 2.11: A complete river system (left) and a subset selected by Strahler order values (right). The inset pictures are at half the scale of the larger pictures.

Network orders assign a level of significance to each river that can help the data simplification of Section 2.4; we can omit less significant rivers as the map scale shrinks. The omissions maintain the visual clarity of the map. For example, Figure 2.11 shows part of the Nechako river system in British Columbia at two different sizes (the inset figure has one quarter the area of the larger figure). The picture on the left shows the river segments with Strahler order at least 2 while the picture on the right only has the river segments with Strahler order at least 5. As the map scale is reduced in the inset figure, the complete river system on the left becomes crowded while the subset of the river on the right still allows us to identify the larger structure of the river.

Each of the network orders assumes that the river network is a directed tree. However, some rivers are better represented as acyclic graphs. For example, a braided river splits and merges with itself but the flow of the river remains in a consistent downstream direction. The network orders do not specify what to do when a river splits or merges with itself.

Figure 2.12: A braided river

One failing of the network orders is their inability to handle polygonal features such as lakes or wide rivers that are modelled by their banks. These cases resemble braided rivers where the orders along one river bank should affect the orders along the opposite bank. In a graph with banks, the topological orders of the network need along opposite banks are independent until the banks merge at their downstream point. We briefly address this problem in Section 3.5.

2.7 Medial Axis, Voronoi Diagram, and Delaunay Triangulation

We review the definitions of geometric structures such as the medial axis, Voronoi diagram, and Delaunay triangulation, their representations in the computer, and the benefits they provide for river analysis.

A polygonal line or *polyline* is defined by a sequence of points p_1, p_2, ..., p_n, called *vertices*, and the line segments, called *edges*, that join consecutive vertices. We assume that the only intersections between these segments at common endpoint. A *polygon* is a circular sequence of edges, which can also be considered as a polyline whose first and last vertices are identical.

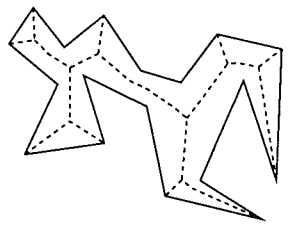


Figure 2.13: The medial axis of a polygon (dashed).

The *medial axis* of a polygon P is the set of centres of circles that are contained inside P and are tangent to two or more different polygon edges [10, 75]. For this purpose, polygon vertices, where two edges meet, count as a single edge. For any such circle, its centre is equidistant to the two tangent edges and is therefore on the *bisector* of the two edges, as illustrated in Figure 2.13. It is natural to associate a point p on the medial axis with the two boundary points that are tangent to the circle centred at p.

Although we focus on the medial axis of a polygon, we consider it as a special form of a

closely-related data structure, the Voronoi diagram. The *Voronoi diagram* [5, 69] for a set of point *sites* $\{x_1, x_2, \ldots, x_m\}$ in the plane is the decomposition of the plane into maximally-connected regions that have the same set of closest sites. (The Voronoi diagram can be defined in the same way for sites that are line segments, as long as no two sites intersect.) There are three types of regions. The *Voronoi cell* of a site x_i contains those points with x_i as the unique closest site. *Voronoi edges*, with two closest sites, are segments of the perpendicular bisector between the sites. Voronoi edges meet at *Voronoi vertices*, which are points that are equidistant to three or more closest sites. The dashed edges of Figure 2.14 show an example of a Voronoi diagram.

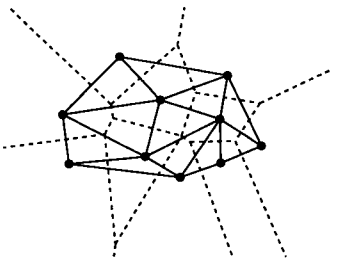


Figure 2.14: The Voronoi diagram (dashed) and Delaunay triangulation (solid) of points.

Figure 2.14 shows another diagram that is defined on the same set of sites. The *Delaunay triangulation* of a set of sites S is the triangulation of S in which every triangle T satisfies the *empty circle property:* the circumcircle of each triangle T does not contain any point of S in its interior. The point s is outside the circumcircle of points p, q, and r (found in this counterclockwise order around the circle) if the following determinant is greater than 0:

$$\begin{vmatrix} 1 & p_x & p_y & p_x^2 + p_y^2 \\ 1 & q_x & q_y & q_x^2 + q_y^2 \\ 1 & r_x & r_y & r_x^2 + r_y^2 \\ 1 & s_x & s_y & s_x^2 + s_y^2 \end{vmatrix}$$

Mathematically, the Voronoi diagram and the Delaunay triangulation are *dual graphs* of one another. Three sites that define a Delaunay triangle T also define a Voronoi vertex at the centre of the circumcircle of T. Two sites define a Voronoi edge if and only if the same two sites are adjacent in the Delaunay triangulation. The adjacencies between faces of the Delaunay triangulation are topologically equivalent to the adjacencies of Voronoi vertices for the same set of sites. Consequently, each diagram contains the same topological information and we can efficiently derive one diagram from the other.

The medial axis of a polygon is a subset of the Voronoi diagram whose sites are the open edges and vertices of the polygon. The edges of both the Voronoi diagram and the medial axis are bisectors between nearest sites. More specifically, the portion of the Voronoi diagram that lies

inside a polygon P contains the medial axis of P; the Voronoi diagram includes Voronoi edges to the reflex (non-convex) vertices of P that the medial axis omits. The medial axis can be derived from the Voronoi diagram in linear time.

Chapter 3

River Centrelines

We can use a river network to answer many questions. For example, given two points, can we navigate from one point to the other through the rivers and, if yes, what is the path? If a stream is contaminated with a pollutant, which other rivers may be affected? What is the sequence of a set of rivers along a common mainstem? Given a school of salmon fish at the mouth of a river, where might they reach as they swim upstream and how many fish might reach each river junction in the network?

This chapter presents an algorithm for automatically finding centrelines of rivers to answer these questions. The challenges are twofold: finding the centrelines in a robust manner and orienting the flow of water along the centrelines. The centreline algorithm also maintains trace-back information to river banks, from which we derive additional characteristics of the river network.

The ease or difficulty of answering these kinds of questions depend on the type of river data. Each of these questions involves the topology of the river network. In the simplest case, only the river centrelines represent the rivers and there are no braided rivers or river deltas. The river network is then a simple graph—a tree—and standard algorithms for finding connected components, and directed and undirected paths in graphs answer the questions.

In a more complex case, some rivers appear as centrelines, other rivers appear as pairs of river banks, and the river network contains lakes. While this data is still a graph, it is not as simple as a tree. An algorithm for components of a graph can still answer questions about whether or not there exist paths between two points, but the path in the graph could trace the outline of a river or a lake instead of just crossing the lake. Questions that ask about upstream and downstream

relations aren't as easily answered. The graph has no information to link rivers that are on opposite banks of the same river. We focus on identifying the topology of the complex river data.

Our motivating application for this work is a system to model the migration of salmon in the Fraser River of British Columbia, Canada. A number of river attributes are correlated with the suitability of the river system for salmon migration and salmon spawning [36]. Some attributes, such as current strength and water temperature, are dynamic and must be monitored throughout the spawning season. Other attributes, such as reachability of rivers, order of rivers along river mainstems, river slope, river length, and river area, are more static and change only over a period of several years.

For the static attributes, past analysis on rivers treats the river network as a directed tree. For a tree edge e in a graph embedding, we can calculate its length, slope, edges reachable by a directed path (edges downstream of e) or reachable in the reversed graph (edges upstream of e), and its connected component. A nominal river width multiplied by the length of an edge e estimates the river area along e. For a pair of tree edges, we can calculate whether one is upstream of another or whether they belong to incomparable branches of the network. The tree also acts as a base structure to which we tie dynamic attributes to edges for a common analysis.

While we generally assume that more-detailed data contains more information, that information is not necessarily accessible. For example, detailed river networks often encode riverbanks and lake shores instead of river centrelines. The earlier directed graph paradigm for analysis becomes less useful since we now represent rivers implicitly by their banks. For any of the static river attributes, we must either choose one of the two banks on which to collect the data or rationalize measurements between opposite banks, which assumes that we know which banks are opposite one another.

When we refer to rivers and river banks in this dissertation, we also handle lakes by treating them as wide, yet possibly short, rivers. The rivers and lakes differ in their geographic interpretation, not in their geometric representation.

If a river system also contains wide rivers, the static river attributes are more difficult to compute, even though wide rivers are visually more descriptive. The problem is that the structure of river systems with wide rivers is more implicit in the coordinates, and less explicit in the data structure, than the structure of systems with narrow rivers alone. For example, to compute area

of a section of river, one can decide on where the section begins and ends for each bank. Opposite banks are likely to have different slopes and lengths. Connectivity and river order of tributaries on opposite river banks are not directly computable from the polyline representation of the banks.

One not uncommon way to automatically generate river centrelines, which we also use, is to compute the *medial axis* of the wide rivers. The medial axis gives a natural way to associate banks and centrelines. Preserving this association—tagging each medial axis edge with the two closest river banks and each river bank with its nearest medial axis edge—lets us preserve the bank information for computing river width and other attributes.

Although the medial axis is a well-defined structure, calculating the structure exactly in the presence of degeneracies can be difficult. This section presents a piecewise-linear approximation to the medial axis; it is a compromise that preserves the desirable properties of the medial axis and is robust enough to handle data degeneracies in river networks. We develop the approximation and supporting functions to handle the characteristics of river data: the data may contain errors and will contain degeneracies. After we have addressed these data constraints, we prove that the approximation still converges to the medial axis and associates river banks with the approximation as a centreline.

This section begins with a description of our source data (Section 3.1) and methods for detecting and correcting errors in the data (Section 3.2). We describe how we compute the approximation from a robust Voronoi diagram algorithm in Section 3.3 and how we orient the medial axis in Section 3.4. Section 3.5 shows that the structure of the medial axis gives a natural identification from banks to centrelines that allows us to

- identify points on opposite banks of a river,
- tie analysis on centreline networks to original river bank data,
- calculate surface areas for rivers, and
- extend network orderings, such as the Horton, Strahler, and Shreve orders on river networks, to include lakes and wide rivers for cartographic generalization.

Section 3.6 then describes methods for computing the surface area of rivers and lakes from the medial axis approximation. We close with a brief evaluation in relation to our motivating application in Section 3.8.

In this section, we handle the river data in two distinct ways. With the first way, we use the semantics of the edges (left bank, right bank, lake shore, etc.) to characterize the data and correct potential errors. With the second way, we do not distinguish between edge types and treats both lakes and rivers as polygons, and often combines lakes and rivers into one big polygon. This latter way only uses the edge directions to define the interior of polygons for computing a medial axis.

3.1 TRIM Data

The data for our salmon migration application was supplied by the Canadian Department of Fisheries and Oceans and Facet Decision Systems. It is a set of coded polylines that outline terrain features. We use the hydrological features: rivers, river banks, and lake shores. The data is digitized at a resolution for 1:20 000 scale maps with a 10 metre accuracy in the xy-plane and a 5 metre accuracy in elevation. Points are specified in UTM coordinates and are rounded to the nearest integer: x values in one UTM zone range between 0 and 10 000 000 and y values range between 0 and 100 000 000. The data adheres to the 1:20 000 TRIM data standard of British Columbia [86]:

- Rivers and river banks are digitized in a downstream direction.
- Lake shores are digitized in a clockwise direction.
- A river whose width is less than 20 metres is digitized as the centreline of the river.
- A river whose width exceeds 20 metres is represented by polylines that are tagged as left and right banks; no explicit association between opposite banks appears in the source data.
- A river r that enters another river or a lake ℓ is separated from that river or lake by two artificial edges called *construction edges*, one to close river r and the other to close ℓ (Figure 3.1). The construction edges are labelled as river banks or lake shores and do not correspond to physically identifiable parts of the river.
- Polylines only meet at their numerically identical endpoints.

The polylines are not annotated with their adjacent edges.

The polygonal lines are unordered, so we must derive and store the topology or adjacency structure of the data before computing the medial axis of the features. Since adjacent lines share exact endpoints, we place all the line endpoints into two-dimensional buckets and use the matching

points within each bucket to define adjacent edges. The matched ends provide enough topology to trace the outline of lakes and rivers. The main digitizing errors found while creating the topology were open polygons, miscoded edges, reversed edges, and missing edges.

We also use TRIM elevation data to find the watersheds of rivers in Section 5. The elevation data has the same accuracy and precision as the rivers: 10 metre accuracy in the xy direction and 5 metre accuracy in elevation with a 1.0 metre precision. However, the TRIM hydrographic features follow clear lines on the terrain and are digitized more accurately than the elevation points.

3.2 Common Digitising Errors in River Networks

Implementations of computational geometry algorithms rely on geometric properties of the data to produce coherent results. Meanwhile, GIS data often contains inconsistencies simply through the volume of data. Consequently, a careful implementation of an algorithm tests its inputs against the algorithm requirements to ensure a logical consistency of the data before proceeding with the algorithm.

In this section, we translate the characteristics of TRIM hydrology data into graph properties and describe errors in the TRIM data that can violate these properties. In some cases, we can automatically correct the errors. This is where the meaning of each edge, as a left bank, right bank, or lake shore, is important.

The TRIM hydrology data defines a directed graph in which each graph edge e has a type attribute that is one of *single river*, *right river bank*, *left river bank*, or *lake shore*. The geographic locations of the rivers give a planar embedding of the graph. The geographic locations of rivers, river banks, and lake shores ensure that these features do not cross in the graph embedding. Only construction edges, which are artificial edges that separate two rivers at their confluence or that separate a lake a river, might cross one another; construction edges have a type attribute of *right river bank*, *left river bank*, or *lake shore*. In an embedding, the construction edges close their respective river or lake polygons only if they do not cross; if edges e and f share endpoints u and v, and if e is the edge immediately clockwise of f around u then f is the edge immediately clockwise or e

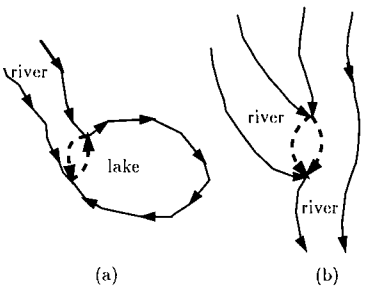


Figure 3.1: Construction edges (dashed lines) where rivers and lakes meet

around v. When a river meets a lake, we can always embed the construction lines at the junction so that the construction edge labelled *lake shore* belongs to the face for the lake and the construction edge for the river belongs to the face for the river (Figure 3.1a). Similarly, when two rivers meet, one river is deemed the mainstem and we can always embed the construction edges so that the river banks of the mainstem remain unbroken (Figure 3.1b). Throughout this dissertation, we always refer to this particular embedding of the river networks to define the faces of the graph.

The data characteristics of Section 3.1 appear as properties of the directed graph relative to its planar embedding. We label these properties as T1 to T5 and D1 to D3 for future reference:

T1 If the boundary of a finite face C contains edges of type *left river bank* and of type *right river bank* then C can be decomposed into two directed paths π_1 and π_2 where all edges of π_1 have type *left river bank* and all edges of π_2 have type *right river bank*.

T2 If some edge e has type *lake shore* then the face to the right of e is a finite face whose boundary is a directed cycle C and all edges of C have type *lake shore*.

T3 If some edge e has type *left river bank* then the face to the right of e is a finite face that only involves edges of type *right river bank* and *left river bank*. The same conclusion holds for the left face of an edge f with type *right river bank*.

T4 At a vertex v,

- let l_v be the number of incident edges of type *left river bank*,
- let r_v be the number of incident edges of type *right river bank*, and

Denote incoming and outgoing edge counts at v with $+$ and $-$ superscripts (respectively). Then, for all vertices v of the graph, $l_v + r_v$ is even, and at least one of the following statements is true:

- rivers begin at a point: $l_v^+ = r_v^+ = 0$, $l_v^- = r_v^-$
- rivers end at a point: $l_v^- = r_v^- = 0$, $l_v^+ = r_v^+$
- construction edges start: $l_v^- = r_v^- = 1$, $l_v^+ = r_v^+$
- r_v^+ rivers end on the left bank of another river: $l_v^- = 1$, $r_v^- = 0$, $l_v^+ = r_v^+ + 1$
- r_v^+ rivers end on the right bank of another river: $r_v^- = 1$, $l_v^- = 0$, $r_v^+ = l_v^+ + 1$

T5 At a vertex v, the number of incident edges of type *lake shore* is even.

Although not mandated by the TRIM specification, any river system in which the rivers have an appreciable slope (where rivers do not form a cycle, as in M.C. Escher's Waterfall [82]) satisfy additional properties:

D1 If the boundary of a finite face is a directed cycle C, then the cycle either has two construction edges or every edge of C hast type *lake shore.*

D2 After removing construction edges and collapsing lakes to a single graph vertex, the directed graph is acyclic (a consequence of property D1).

D3 If neither end of an edge e is a leaf, if e has type *single river*, and if there are edges of type *single river* adjacent to both endpoints of e then there exist edges d and f adjacent to the endpoints of e where def is a directed path in the graph.

In the following subsections, we use these properties to automatically filter our data. During our experiments with geographic data the filters were invaluable in obtaining semantically-consistent data for our algorithms. Lemmas 3.2.1 and 3.2.2 show that our filters enforce a consistent embedding of the river data.

3.2.1 Data Degeneracies

When polylines are digitized on a map, it is not unusual to obtain two consecutive points that are identical if the mouse input is not debounced. Consecutive identical points can also occur when the precision of the data is decreased and two consecutive points are rounded to the same location.

We can detect and automatically correct these errors with a linear scan of each polyline. The scan determines whether or not adjacent points are identical and, if so, remove the duplicate point. All the data simplification algorithms of Section 2.4 except for the nth point selection algorithm can remove duplicate adjacent points.

3.2.2 Intra-face Semantics

Property T1 specifies how the edges can meet one another at a face.

It is feasible to test property T1 by finding every finite cycle in the network graph that satisfies its precondition and testing for its conclusion, but we opt for a local test that is suggested by the results of Lemma 3.2.1. We test the necessary condition that river bank edges e and f in a common face meet head-to-tail only if they have the same edge type.

Lemma 3.2.1 *Given a face of the network that satisfies property T1 and two river bank edges e and f adjacent on the boundary of the face, edges e and f meet head-to-tail only if edges e and f have the same edge type.*

Proof:

The directed paths π_1 and π_2 of the conclusion to T1 guarantee the conclusion of the lemma inside the paths.

Let e be the end of π_1 and be adjacent to edge f on π_2 at its head vertex v. The symmetric argument applies at the start of the first edge for π_1. Edge e of type *left river bank* has a river r_1 to its right and edge f of type *right river bank* has a river r_2 to its left. Assume that e and f meet head-to-tail and derive a contradiction.

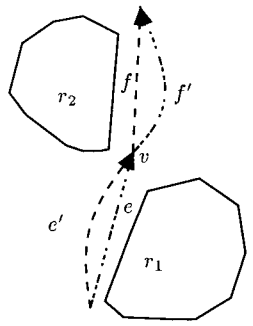


Figure 3.2: Construction edges of Lemma 3.2.1

If $r_1 = r_2$ then river r_1 crosses the bank at v and r_1 appears at both the right and left sides of e. Therefore, edge e is not a left river bank — a contradiction.

If $r_1 \neq r_2$ then either e or f has r_1 and r_2 on both sides and is a construction edge. Without loss of generality, assume that edge e is the construction edge. Then there is a matching construction edge e' of type *right river bank* adjacent to f at v and, by the embedding of the river, e' separates e from r_2 (Figure 3.2). Then the common face between e and f must contain r_1, so river bank f has a river on each side and is also a construction edge with a matching *left river bank* edge f'. By the river embedding, f' separates f from r_1; edges e and f no longer share a face—a contradiction.

Therefore edges e and f must meet tail-to-tail or head-to-head. ∎

The local test is that left and right bank edges for one face can only meet tail-to-tail or head-to-head at a vertex. A direct consequence of property T1 is that two banks of the same type can only meet tail-to-head.

There are several explanations for data that fails this local test. We encountered three possibilities:

1. An edge is mislabelled.
2. An edge is reversed.
3. A construction edge is missing.

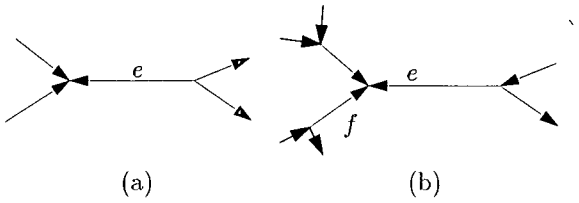


Figure 3.3: A reversed edge e that can (a) and cannot (b) be corrected automatically.

Since there is more than one possible explanation, we can detect these errors but can only correct them in cases where there is more contextual information.

For reversed edges, if an edge only has incoming edges at its head and outgoing edges at its tail then it is incorrectly oriented and can be reversed automatically (Figure 3.3a) from property D3. Otherwise, a vertex p in the river network graph that has only incoming edges or has only outgoing edges probably has one of its adjacent edges incorrectly oriented (one of edges e or f in Figure 3.3b). Reversing the direction of either e or f leads to a consistent flow in the network. Since we do not have other cues to indicate which edge to reverse, our diagnostic software highlights all the edges around p for the user to consider.

For missing construction edges that fail to close a river or lake, such as at a map boundary, we can start at one open edge of the river polygon and trace the polygon until we reach an unpaired river edge. We create a straight-line construction edge between these two open edges to close the polygon automatically.

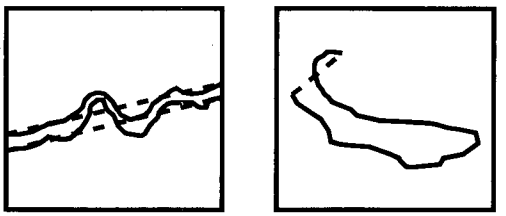


Figure 3.4: Incorrect polygon closures across a map sheet (a) and across obstructed endpoints (b)

The closure fails in two cases. First, if the polygon extends completely across a map sheet then tracing the polygon face leads from one side of the map sheet to the other (eg. from the left side of the sheet to the right side of the sheet in Figure 3.4a). The boundary of the map sheet is not an actual line that closes the polygons. In this case, the construction edge incorrectly cuts across the map. Second, if the endpoints of the two open edges are not mutually visible then the construction edge produces a polygon that crosses itself (Figure 3.4b). This latter failure cannot occur at map sheet boundaries, but can occur if the user selects a subset

of the river network for analysis and relies on automatic corrections to make the subset conform to the TRIM standard. Both of the failure cases require user intervention to detect and resolve.

3.2.3 Edge Adjacency Semantics

If some edge e has type *lake shore* then the face to the right of e is a finite face whose boundary is a directed cycle. This relates to properties T2 and T3.

Tests for these conditions can fail in three ways. First, the face may not be finite, in which case we have a lake or river that must be closed (manually or automatically) with a construction line as previously described.

Second, the edge in the face may be incorrectly oriented. We can confirm this error by examining the face to the left of e if e is a *lake shore* or *right bank*, to the right if e is a *left bank*. If the face does not contain any edges and the other face does contain edges (such as rivers) then the directions of the boundary edges around the face are incorrect; the lake is digitized in a counterclockwise direction. Alternatively, we can compute the signed area of the face where only a face that is traversed in a counterclockwise direction yields a positive area value; a positive area value for a lake indicates that the lake is digitized in a counterclockwise direction. We can automatically reverse the edges in such a case.

Third, some edge may extend into the lake. This can occur when a single-river meets a lake and is extended into the lake (Figure 3.5). Digitising software can break edges where they cross, leaving one polygonal line inside the lake. These extensions are semantically confusing since lakes cannot contain rivers. As long as we can confirm that the lake is properly oriented, we can remove the river extension automatically and safely.

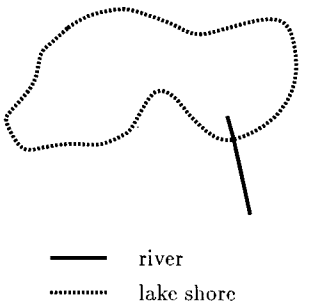


Figure 3.5: A river segment extended into a lake.

3.2.4 Inter-face Semantics

Property T1 describes how edges in one face can meet; properties T4 and T5 complement condition T1 by describing how many edges can meet at a vertex of the network graph. The properties imply a local parity check on the edge types at each vertex.

As an automatic test condition, the reasons for failure and the possible automatic solutions

are the same as for property T1. While property T1 finds errors in one face, T4 and T5 find errors between adjacent faces.

The combination of properties T4 and T5 of Section 3.2 and the necessary condition of T1 in Lemma 3.2.1 imply property T1 of Section 3.2

Lemma 3.2.2 *Assume that every pair of adjacent river banks e and f for each river meet head-to-tail only if they have the same types of river banks. If the standard embedding of the river graph satisfies property T3 then the river graph is missing pairs of construction edges or satisfies property T1.*

Proof: Let C be a face of the river graph with edges of type *left bank* and *right bank*. From property T3, all edges of C are of type *left bank* and *right bank*. Since adjacent edges e and f meet head-to-tail only if they have the same bank type, the boundary of C decomposes into maximal directed chains of a common type, chains $\ell_1, \ell_2, \ldots, \ell_n$ of type *left bank* and chains $r_1, r_2, \ldots, r_m$ of type *right bank*, and the chains alternate in type and orientation around C. Assume that all pairs of construction edges occur in the river graph; if only one edge of a pair existed then one face would violate our head-to-tail assumption. We will conclude that property T1 must be satisfied.

With the construction edges, the definition of when construction edges are added to a river imply that each river has a single drain point. The river bank edge semantics imply that when r_i and ℓ_j meet head-to-head then this is a point through which the river C can drain. As long as the construction edges exist, each face has exactly one drain point. The ordering of construction edges in the standard embedding of the river graph ensures that the edge type for a construction edge in a river matches the type of its upstream edge on the same face. Therefore n and m are 1 and we have two chains in the boundary of C. ∎

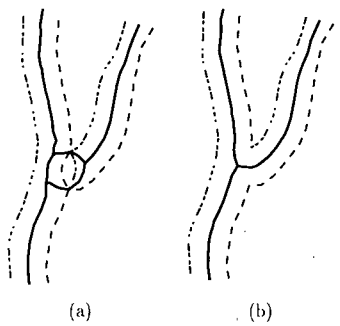


Figure 3.6: Bad (a) and good (b) medial axes around construction edges

It is natural to perform the local tests for T1, T4, and T5 in the same scan of the river graph. However, it is more convenient to find single-face errors with the necessary condition of T1,

remove all construction edges from the river graph, and then apply the test for condition T4 to locate subsequent inconsistencies. Removing the construction edges merges adjacent faces, typically two rivers or a river and a lake, which by property D2 leaves an acyclic graph. Keeping in the construction edges when finding the medial axis leads to branches around the construction edges in the final river centreline (solid line in Figure 3.6). By applying the test for condition T4 after construction edges are gone, we still ensure that the lake and river faces are closed, even after they are combined.

3.3 Medial Axis Approximation Algorithm

The medial axis is a well-studied structure in image analysis and in computational geometry. Image analysis algorithms typically discretize the problem with a rectangular grid of pixels and compute which pixels are (approximately) in the medial axis [30]. This is not a good fit with our input vector data for rivers: thin, meandering polygons defined by sequences of unevenly scattered vertices. Neither is it a fit with the output we desire to exploit: topological connections within the medial axis and back to the original data.

Computational geometry has developed optimal algorithms to compute the structure for simple polygons [1, 19]. The medial axis is also a subset of the Voronoi diagram of the polygon edges and, as such, algorithms that compute Voronoi diagrams [11, 37, 43, 69, 75, 97] or constrained Delaunay triangulations [18, 27, 79] can find the medial axis.

Unfortunately, implementations of algorithms for the medial axis or the Voronoi diagram for polygons [2] must deal with the parabolic segments in the diagrams. The more successful implementations use incremental constructions and topological information [84] or triangulations [43] to resolve degenerate conditions and to avoid relying on tests with parabolic segments to make decisions. While good in practice, incremental algorithms have non-optimal worst-case behaviours. Consequently, we adapt a robust sweep algorithm for the Voronoi diagram of point sites to approximate the medial axis. The sweep algorithm provides a better worst-case time complexity. The restriction to point sites also means that we need not handle parabolic segments in the final diagram. Moreover, an implementation for the Voronoi algorithm for point sites has fewer types of degeneracies to deal with than a comparable implementation for line segments. The Voronoi diagram of a discretized polygon boundary produces a matching Delaunay triangulation for the

interior of the polygon and this triangulation guides our approximation to the medial axis.

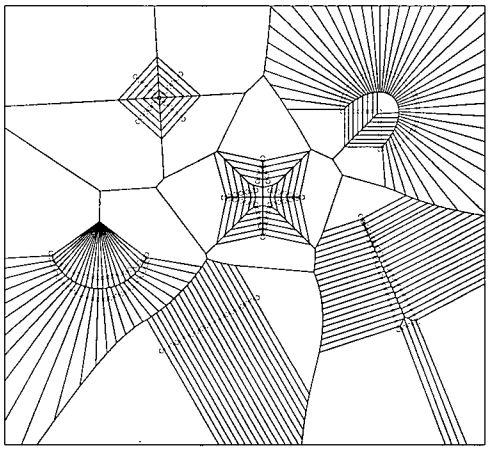


Figure 3.7: The Voronoi diagram of lines and curves from the Voronoi diagram for points

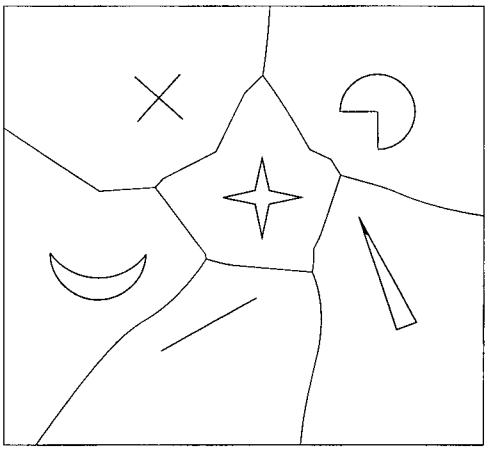


Figure 3.8: The Voronoi diagram of lines and curves

As the cover of Okabe, Boots, and Sugihara [69] demonstrates (reproduced in Figure 3.7), we can approximate the Voronoi diagram of lines or curves with the Voronoi diagram of points along the lines or curves boundaries. Figure 3.8 shows the Voronoi diagram of lines and curves that Figure 3.7 approximates. By spreading more points along the lines or curves, the Voronoi diagram of the points converges to a superset of the Voronoi diagram for the lines or curves. Thus, given the polygonal contour of a river or lake, we will discretize the boundary of the river, compute the Voronoi diagram of these points, and approximate the medial axis from the result.

Discretization always involves trade-offs. The quality of the approximation depends on the degree of discretization for the boundary. As more points discretize the boundary of the polygon, the Voronoi diagram inside the polygon converges to a superset of the medial axis for the polygon. Computationally, adding more points to the boundary adds degeneracies and increases the computation time. We must strike a balance between computation time and fidelity of the approximation.

Our solution adaptively discretizes the river boundary until the Delaunay triangulation of the discrete boundary contains all river boundaries. Unlike Nackman and Srinivasan [64], we do not have the medial axis to guide our discretization. Since hydrology data is usually well-sampled,

```
Q ← {p_0, p_1, ..., p_{n-1}}
D ← the Delaunay triangulation Q
for i = 0 to n − 1 do
    /* Assert that P from p_0 to p_i is in D */
    push p_i onto S
    while S is not empty do
        q ← pop(S)
        if segment from q to q.next crosses a Delaunay triangle then
            t ← midpoint of segment from q to q.next
            Q ← Q ∪ {t}
            update D
            push the neighbours of t in D onto S.
        endif
        repeat the if statement with the segment from q to q.prev
    endwhile
endfor
```

Algorithm 1: Algorithm to discretize a polygon boundary

we want to avoid splitting every edge as with Saalfeld's algorithm [78].

We treat both rivers and lakes as polygons when approximating the medial axis. In fact, we can eliminate construction edges in the river networks to combine rivers and lakes into larger polygons for the medial axis computation. We only revert to using the type of the edges when we orient the medial axis in Section 3.4.

We start with three data structures:

- A doubly-linked list of the vertices for the polygon P.
- A quad-edge storage structure for a triangulation.
- A stack S of points on the polygon P.

The n points of the polygon P are $p_0, p_1, \ldots, p_{n-1}$ and will always be referred to as such, even after we subdivide edges of P. The linked list of points for P has a pointer into the quad-edge storage structure so that each point references one edge for which it acts as an endpoint. The quad-edge structure also has a references from each edge to its origin vertex in P.

Figure 3.3 presents the incremental algorithm.

The algorithm incrementally updates the current triangulation [85] and compares the new Delaunay triangles with the river boundary again. We divide river boundary edges until no Delau-

nay triangle crosses the river boundary. Upon completion, the Delaunay triangulation decomposes the river interior into triangles (Figure 3.9) and the corresponding Voronoi edges inside the river form a single component.

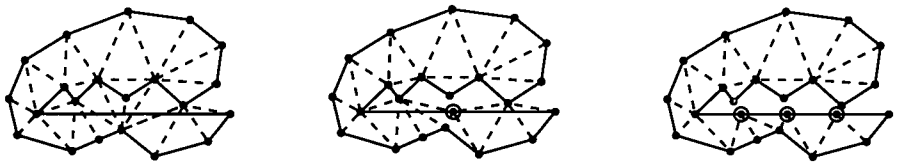


Figure 3.9: Delaunay triangles inside the polygon with the initial points (a), and after one (b) and two (c) decomposition steps

Note that discretization by adding points on the edges of a polygon is different from discretization of the whole space as in image processing. The former is more adaptive and does not lose the topology of the output.

Lemma 3.3.1 *Our algorithm for computing the discretized Voronoi diagram of n lines takes $O(n \log n + k(n+k))$ worst case time and $O(n \log n + k)$ expected time where k points are added to the edges.*

Proof: The initial Delaunay triangulation of the n line endpoints takes $O(n \log n)$ time. The for loop iterates n times, but we will count its work with points inserted into Delaunay triangulation. For any point on the stack, we locate the wedge that contains its edge and test for the edge in D in $O(n)$ worst-case time and $O(1)$ expected time. The cost of the test goes to the point itself for the first test on p_i and subsequently on the point whose insertion placed q onto the stack S. To add the ith incremental point q, we first split the edge and update Q in constant time, insert q into the triangulation D in worst case $O(n+i-1)$ time and expected $O(1)$ time and push at most $O(n)$ (and expect $O(1)$ points onto the stack S. The worst-case time complexity after inserting k points is then $O(n \log n+n+k(n+k)) = O(n \log n+k(n+k))$ and expected time complexity of $O(n \log n + n + k) = O(n \log n + k)$. ∎

Figure 3.10 shows an example by Edelsbrunner and Tan [34] with n vertices and m lines that requires $\Omega(nm)$ additional points if the line segments will appear as Delaunay edges. For lakes and river polygons, $m = n$ so our algorithm, as stated, has an $O(n^4)$ worst-case time complexity and an $O(n^2)$ expected time complexity. These are intolerable complexities for GIS applications if they represent the normal behaviour. If these lower bounds were representative of the running time on GIS data then the algorithm would be too slow to be practical.

Figure 3.10: A lower bound for line subdivisions.

The worst-case scenario, where polygons fold on themselves to produce the configuration of Figure 3.10, are improbable in river data. Even for a serpentine river, we need many digitized points along the river banks to capture the curvature of the river. The main locations for edge refinements are in bulges in the river banks and in narrow inlets to a river. Consequently, the number of points added to the river polygon boundary is proportional to the size of the polygon and the incremental algorithm has an $O(n^2)$ worse-case time behaviour and an $O(n \log n)$ expected time behaviour.

We have three approximations to the medial axis centreline that come from the Delaunay triangulation of the discrete boundary. The three approximations are different embeddings of an abstraction of the medial axis. Let G be the graph dual of the Delaunay triangulation D for the interior of polygon P; exclude the vertex for the outer face of P in G. Then G is a connected planar graph. Our approximation is a subgraph of G that connects all tributaries to P.

For each tributary of P, designate an adjacent triangle of D as the initial flow direction for the tributary into P. These triangles correspond to a set of vertices T in G. Root G at one vertex t of T. There exists a unique path $\pi_t(s)$ in G from every vertex s in T to t. The medial axis approximation A is the subgraph of G induced by these paths: $A = \bigcup_{s \in T} \pi_t(s)$.

If P has only one tributary, then this approximation contains only the root of the tree. It does not provide any detail on the shape of the lake. To extend the approximation beyond the root, we add to T the vertex u for the Delaunay triangle farthest away from the tributary. Vertex u extends the approximation from this root so that it captures part of the shape for the lake.

This medial axis approximation is an abstract graph that gives the interconnections between tributaries. To see the approximation, we must embed the graph. We consider three straight-line

embeddings for A by selecting different positions for a vertex v of A that corresponds to a Delaunay triangle d of D:

- **Voronoi approximation**: embed v at the Voronoi vertex for d—the centre of the circumcircle for d (Figure 3.11a).
- **centroid approximation**: embed v at $\frac{a+b+c}{3}$ where a, b, and c are the vertices of d (Figure 3.11b).
- **midpoint line approximation**: embed v at $\frac{a+b+2c}{4}$ where a, b, and c are the vertices of d and segment $\overline{ab}$ is the shortest edge of d (Figure 3.11c).

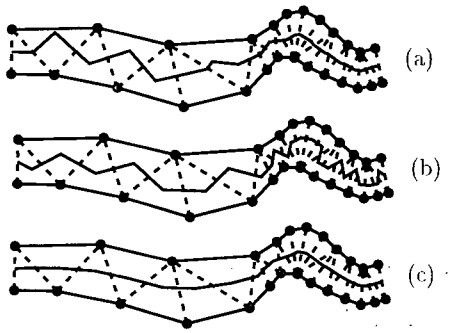


Figure 3.11: The Voronoi (a), centroid (b), and midpoint (c) approximations to the medial axis.

In the Voronoi approximation, the edges remain inside the river as long as the incremental discretizing forces the triangle angle opposite a river edge to be smaller than 90 degrees. Unfortunately, if the discretized points along a boundary edge are far from one another (relative to the river width) then this approximation has a zig-zag pattern rather than the expected smooth centreline.

In the centroid approximation, the centroid is a natural choice as a representative point for a triangle, but the paths between centroids is not smooth. When the bases of the marked triangles alternate between river banks, the centroid line approximation zig-zags again. The zig-zag is most pronounced when the triangles have one side much smaller than the other two sides.

The midpoint line approximation is similar to the centroid line. Geometrically, if p is the midpoint of line segment $\overline{ab}$ then the point $\frac{a+b+2c}{4}$ is the midpoint of the line segment from c to p; this point is always inside triangle abc. With a fine boundary discretization, the shortest edge of the Delaunay triangles are usually along the river banks so line segment $\overline{cp}$ crosses between river banks.

While all three approximations generate a centreline that lies between the river banks, they differ in two respects: convergence to the medial axis and estimation of river area. Let the distance between two adjacent points along the discretized boundary be at most d. Then let V_d, C_d, and L_d be the Voronoi, centroid, and midpoint line approximations to the medial axis. Lemmas 3.3.2

and 3.3.3 prove that each of V_d and L_d converge to the medial axis as $d \to 0$ and are therefore appropriate for approximating the medial axis. Observation 3.3.4 shows an example of why the lack of convergence of C_d makes it an inappropriate approximation.

Lemma 3.3.2 *The Voronoi approximation V_d converges to a superset of the medial axis as $d \to 0$.*

Proof: Let p be a point on the medial axis equidistant to points q_1, q_2, ..., q_n on the boundary of polygon P. If any two or more points of q_1, q_2, ..., q_n are sites for V_d then p appears in V_d as the bisector of these sites.

Otherwise, let r_1 and r_2 be the neighbours of q_1 and let r_3 and r_4 be the neighbours of q_2 on the discrete boundary of P (Figure 3.12). As $d \to 0$, points r_1 and r_2 both converge to q_1, as points r_3 and r_4 both converge to q_2. The Voronoi point of r_1, r_2, and one of r_3 and r_4 then converges to p in V_d. ■

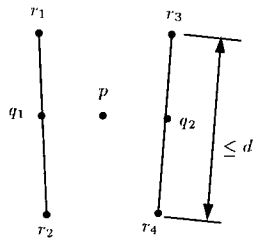


Figure 3.12: Point configuration for Lemma 3.3.2

Lemma 3.3.3 *Let T be the set of Delaunay triangles inside a polygon P that correspond to junctions in the medial axis of P. Let M be the medial axis of P. Then a subset of the midpoint line approximation, namely $L_d \cap \overline{T}$, converges to $M \cap \overline{T}$ as $d \to 0$.*

Proof: Assume that d is small enough so that the boundary of P appears as edges in the Delaunay triangulation.

Let p be a point on the medial axis of P that appears in a Delaunay triangle t not in T. As $d \to 0$, the corners of the shorter edge of t converge to one another and t converges to a shortest chord between opposite boundary points of P. The weighting factors for the representative point of t weigh the ends of the chord equally so the representative point of t converges to p in L_d.

If t is in T then t is not subdivided as $d \to 0$ so there is no convergence to M in any triangle of T. ■

Observation 3.3.4 *C_d diverges from the medial axis as $d \to 0$. Moreover, the length of C_d can be unbounded (Figure 3.13).*

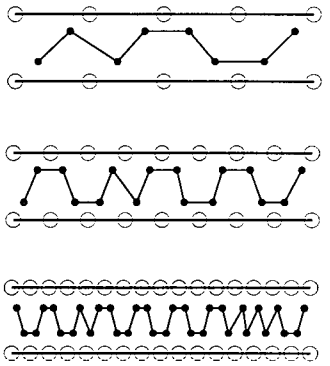


Figure 3.13: Edge subdivision makes the centroid line long.

With successively finer refinements, the centroid line approximation can oscillate between opposite sides of the polygon. The oscillation has an unbounded length as d approaches 0.

In Section 3.6 we describe how the river area is derived from each approximation; in short, the Voronoi approximation samples the river width to estimate the area whereas the other two approximations use the Delaunay triangulation from which they are defined to compute the river area in a simpler and more direct manner. We use the midpoint line in our work as a good compromise between convergence and ease of area estimation.

3.4 Edge Orientation

The medial axis itself solves the river connectivity problem, but does not provide upstream and downstream relations among rivers; we have ignored the direction of water flow along the medial axis edges so far. A correct direction of flow is important to answer queries such as "What is the river area upstream from a particular point?" or "If a particular tributary is polluted, what (downstream) rivers may be affected?"

For rivers, the direction of flow for a medial axis edge matches either the flow direction of a tributary at one end of the edge or the flow direction on the river banks that define the medial edge. This does not apply to medial axis edges inside lakes since the lake shore edges do not encode any flow information. For lakes, a few simple topological rules suffice in our tests for mountaneous regions. The rules essentially enforce the conservation of flow by preventing water from accumulating at any vertex of the medial axis. In a non-degenerate medial axis and in the midpoint line approximation, every junction vertex involves three edges. The conservation argument says that the edges will split as either two incoming edges and one outgoing

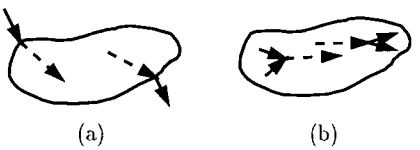


Figure 3.14: Forced flow (dashed) edges of the medial axis

edge or one incoming edge and two outgoing edges. When we have identified two incoming or two outgoing edges at a junction, the direction of the third edge is thus determined. If a junction has three outgoing edges then there are many outflows for the lake; Lemma 3.4.1 characterises this condition.

The rules, in order of precedence, are

1. if an edge is adjacent to a tributary then the flow of the edge matches the flow of the tributary (Figure 3.14a).
2. if the medial axis edges inside a lake meet at a point then there must be at least one edge that enters and at least one edge that leaves the point (Figure 3.14b).

If the medial axis is treated as a tree, then rule 1 orients every leaf edge and rule 2 propagates the edge orientations from the leaf edges to the rest of the tree edges according to a topological order of the tree.

When the medial axis combines the incoming and outgoing tributaries of a lake in a special way, there may not be a unique orientation of the medial axis that conserves flow. Consequently, these two rules are not sufficient to orient all medial axes. Lemma 3.4.1 characterises the instances where the orientation is not unique.

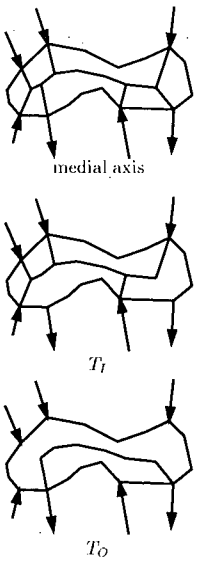


Figure 3.15: The medial axis and flow subtrees

Lemma 3.4.1 *For any set of vertices S of the medial axis M let T_S be the subtree of the medial axis that connects the vertices of S. Let I be the set of leaves of M that attach to incoming tributaries. Let O be the set of leaves of M that attach to the outgoing tributaries. Then M can be uniquely oriented to conserve flow if and only if $T_I \cap T_O$ does not contain an edge of M.*

Proof: Assume that T_I and T_O have at least one vertex each.

Suppose that $T_I \cap T_O$ contains no edges. Then $M \setminus \{T_I \cup T_O\}$ is a path π. To drain T_I, all edges of T_I must be oriented towards π and π must be oriented from T_I to T_O. To satisfy the conservation property at every vertex of T_O, the edges must be oriented away from π, otherwise some vertex does not have an incoming edge. This provides a unique orientation for M.

Now, suppose that e is an edge of $T_I \cap T_O$. Then each subtree of $M \setminus e$ contains both an incoming and outgoing tributary. Inductively, each of these trees can be oriented with a conserving flow. Either orientation of e leads to a conserving flow of M so the orientation is not unique. ■

When the propagation of rule 2 fails because of Lemma 3.4.1, we are left with a set of trees F of unoriented medial axis edges where every leaf in F has one incoming edge and one outgoing edge already oriented in the medial axis. The user can assign directions to one or more of the leaves in F and allow rule 2 to continue propagating. Alternatively, we can iteratively assign an arbitrary direction to one leaf in each tree of F and propagate rule 2. Any such assignment yields a consistent flow on the medial axis, though such a consistent flow may divert the bulk of the lake out a small and relatively unimportant outlet of the lake.

These edge-orienting rules do not detect lakes that have either no incoming edges or no outgoing edges. Neither of these cases exhibit the subtree intersection property of Lemma 3.4.1. In fact, rule 2 completely orients the medial axis edges of these lakes in a self-consistent manner and routes the flow into or out of one centre node of the medial axis. For lakes with no outgoing edges, one consistent orientation is as good as any other: all the water remains in the lake. For lakes with no incoming edges, a consistent orientation can have the bulk of the lake area draining out of a minor tributary.

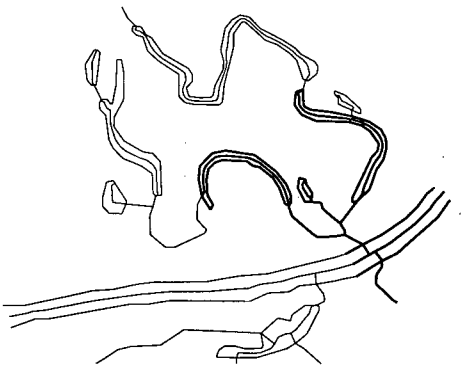


Figure 3.16: Centerline paths with matching banks

3.5 Benefits of the Medial Axis

The main benefit of the medial axis is, of course, to provide a centreline. We derive benefits beyond the centreline from the association between a medial axis edge and its nearest river bank. This section outlines four benefits: the link between derived data and original data, a definition of opposite banks, estimates for river width and surface area, and network orders for river banks and lake shores.

First, the association ties calculations on the centreline to the source data that defined the

centreline. The medial axis replaces the river banks in a network to give a single-line river network that undergoes further analysis, such as identifying drainage basins, locating fish spawning habitat, and tracking the run-off of forest cut blocks. River banks inherit attributes and results of the analysis from their associated medial axis edges. Thus, the single-line network allows for simpler network analysis than on river banks but does not lose the connection to river banks. Figure 3.16 shows a fish migration path in bold computed from the centreline of the river system. The path is tied back to the nearby river banks and lake shores; habitat analysis can then be performed from the bank characteristics.

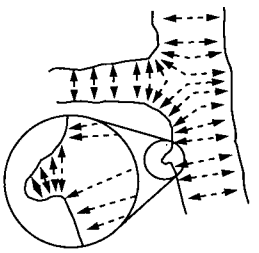
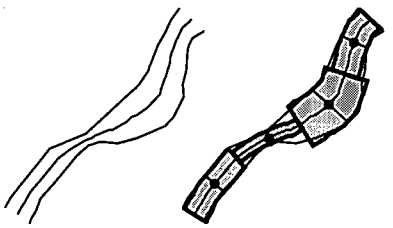


Figure 3.17: Opposite bank relations

Second, we can define opposite banks through the association. Since a medial axis edge is the bisector of its closest river banks, we consider these two banks as opposite one another along the river. We can then combine the attributes of opposite banks, such as slope, elevation, soil type, and vegetation type, into attributes for the centreline or can compare attributes with one another to detect inconsistencies in the data or anomalies in the environment.

Some additional refinement to this simple definition may be necessary, since the medial axis is sensitive to bulges in the boundary of rivers. Within a bulge, the medial axis may identify opposite sides of the bulge as opposite banks, as illustrated in Figure 3.17, rather than identifying the sides of the bulge to an edge across the bulk of the river. This is correct behaviour if there is a tributary that flows into the bulge, but may be inappropriate for some applications when there is no tributary. We discuss this further in Section 3.6.



(a) (b)

Figure 3.18: A river, its centreline (a), and its area estimate (b).

Third, once we use the association to define points on opposite banks of a river, we can use these points to define the width of the river at a point or to define the section of a polygonal river that corresponds to a segment of the single-line network. We define the width of a river at a point p to be the length of the line segment that touches opposite river banks; the medial axis identifies these opposite points on river banks at p. Instead of using one nominal river width for the whole river, we can sample the river width anywhere.

With the river width, we have two approaches to estimate the surface area of a river. We can refine the approach of single-line rivers: partition the length of the river into short sections, sample a nominal width for each section as in Figure 3.18, and sum the area estimates of length times nominal width. Alternatively, we can close off a section of a double-line river, and compute the area of the resulting polygon. Opposite points along river banks are natural candidates for closing off the river. This second approach has a nice implementation that relies on the Delaunay triangulation of the river (see Section 3.6).

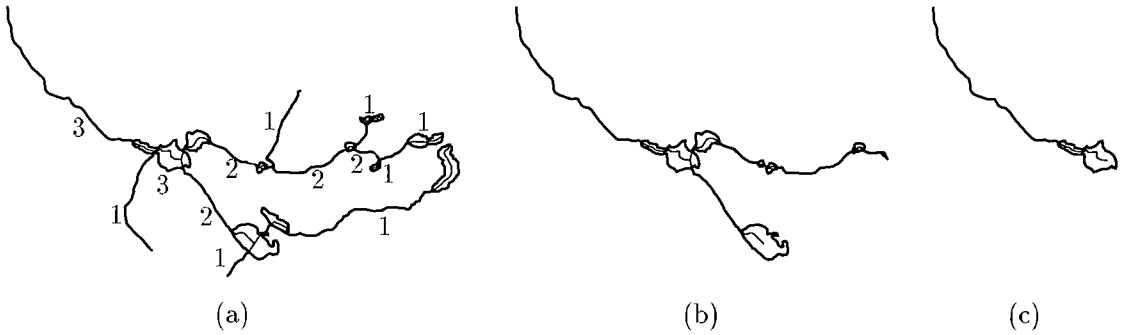


Figure 3.19: A river network with its Strahler order (a) and the subnetworks with Strahler numbers greater than 1 (b) and 2 (c).

Fourth, the association extends network orderings to wide rivers and lakes for cartographic generalization. The medial axis or centrelines of rivers have, for many years, been used as a replacement for double rivers for display at larger map scales [67].

As mentioned in Section 2.6, network orders on single-line networks, such as the Horton, Strahler, and Shreve orders, have been used to select the mainstems of a river network for generalization and display at large map scales. Figure 3.19 shows a sample network with its Strahler order and the result of selecting edges of high order from the network. These same orderings can apply to networks that contain lakes or river banks by treating the lakes and wide rivers as their medial axes; this is not surprising. The river bank edges receive an order number from the associated medial edges. Then selecting edges with higher order also extracts the relevant rivers along the path.

There are two ways to propagate network orders to lake shores and river banks. The first way assigns the network order of the nearest medial edge to a river bank. The nearest edge assignment is appropriate for ordering river banks since the order of the river may increase as we near the

mouth of the river. For lakes, one may prefer to assign the network order of the highest medial edge in the lake to all the lake shores. This assignment treats the lake as a single unit and preserves the visual cues of lake extent and shape.

3.6 Area Generation

The Voronoi approximation to the medial axis provides river width measurements for finding the river surface area with the technique of Section 3.5: partition the river into sections and sample one width value for each river section. The section lengths and sampled widths represent a better area estimate for the sections than the area from a single nominal river width. Equivalently, if the sections all have the same length then the average of the sampled widths yield an overall width estimate for the river. Two drawbacks of sampling the river width to estimate river area are that the samples can miss small bays and can overestimate the width at river junctions.

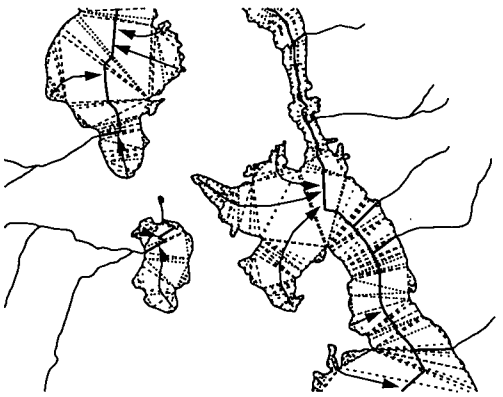


Figure 3.20: The area of bays accumulate in nearby medial edges.

The centroid line and midpoint line approximations to the medial axis can estimate the river surface areas in two ways. The first way is identical to the method of the Voronoi approximation since both approximations capture the same proximity information. The second way assigns the area of each Delaunay triangle to its representative point (centroid or midpoint). Since the Delaunay triangulation decomposes the interior of the river, the representative points account for all the river area. The river area between two points a and b is the sum of the areas assigned to the representative points between a and b along the medial axis approximation. The second way is not directly applicable for the Voronoi approximation since Voronoi vertices do not necessarily lie inside their corresponding Delaunay triangles.

Computing river area from Delaunay triangles has minor drawbacks of its own. First, not every Delaunay triangle has its representative point in the approximate medial axis since the approximation only keeps the triangles that join tributaries. To overcome this drawback, we allocate the areas of inlets and bays that have no tributaries to a nearby representative point to preserve all

the area for the feature. This assignment is shown in Figure 3.20. representative point to preserve all of the area for the feature. Second, the granularity of the area estimate depends on the relative sizes of the Delaunay triangles. The area of a triangle in a river branch may be small while the area of a triangle at a river junction may be large. The granularity only becomes an issue for the ends of river sections.

3.7 Implementation Lessons

We implemented our medial axis approximation algorithm as a library for the Cause & Effect GIS of Facet Decision Systems of Vancouver. The implementation highlighted five issues of merging computational geometry algorithms with a GIS.

First, a user should always have the ability to check (and re-check) the input data and data structures for consistency. With the large amounts of GIS data, there is always an error in raw input data for some algorithm. One type of error is semantic inconsistency as described in Section 3.2. Another error is a bad combination of data, such as lines that cross a map boundary and the maps do not align perfectly so the lines do not meet at a common point on the boundary. Still another error is well-intentioned simplification that skews the topology of the input data. For example, Figure 3.21 displays two simplified polygons that meet at a vertex and whose outer face (dashed line) is topologically merged with an inner face. The sanity-checks that we applied to our input data before running our algorithms saved us from many debugging headaches. Functions that verify the current content of a data structure against invariants for the data structure, called audit functions, reduced the amount of debugging required in the implementation. Moreover, a liberal use of audit functions finds errors in the data structure when the error is created instead of when the erroneous data is later used. Once the debugging of an implementation is complete, we can remove the audit functions to speed-up the execution time.

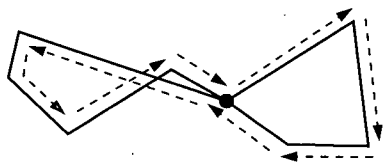


Figure 3.21: Incorrect polygon topology (solid) and the resulting incorrect face (dashed)

Second, we must be prepared for inconsistencies from floating point errors. Our implementation used epsilon geometry [45] on near-zero values and simulation of simplicity [33] on vertical edges in a sweep algorithm to avoid floating point inconsistencies. Despite these efforts, there are still cases where our computed results produced topological inconsistencies, such as merging the

inner and outer faces of a polygon. Letting topological tests guide insertions into data structures, similar to the topological algorithms by Sugihara and Iri for finding Voronoi diagrams of many sites [84], stabilised these inconsistencies and let the data structures adhere to an invariant.

Third, the rule-of-thumb for implementations that states "be lenient in the input data that you accept and concise in the results that you produce" can be taken too far. When applied, the rule-of-thumb simplifies the task of using the output of one program as the input for another. Our implementation of this rule was the ability to relax the requirement that end points be identical to connect two river polylines; given a user-specified tolerance ϵ, any two points that were within ϵ of one another were considered to be the same point. This flexibility was very useful in combining data that came from different maps and whose endpoints did not match perfectly at the boundary. A consequence of this flexibility is that we treat the endpoints of an edge of length less than ϵ as one point, which then creates topological inconsistencies in data structures that track the adjacencies of polylines.

Fourth, we must avoid the common error of reporting results with too much precision. Our implementation made this error when approximating the medial axis. The TRIM data at our disposal has a 1.0 metre precision. The vertices of the medial axis approximation have the precision of a single-precision floating point number. To maintain a consistent level of precision, we should post-process the medial axis to decrease the precision; however, this process must ensure that the reduced precision does not permit the medial axis approximation to cross its river and lake boundaries.

Fifth, non-optimal algorithms can be sufficient for GIS problems. The worst-case (or even expected case) time complexity for our medial-axis approximation algorithm does not measure favourably against the optimal algorithms for medial axes or for constrained Delaunay triangulations. Nevertheless, the simplicity of the algorithm, especially in dealing with data degeneracies, outweighs the implementation complexity.

3.8 Sample River Systems

Our initial tests extracted and directed the centrelines and computed the areas of rivers and lakes in the mountainous interior of British Columbia where lakes have few out-flowing rivers. In the 500 lakes and rivers tested as part of the Horsefly river system, the resulting water flow directions

matched the expected flow directions in all of the cases. In the majority of the cases, the lakes only had one outlet and one inlet so there was a single direction of flow to find. Other rivers or lakes, as in Figure 3.22, have a medial axis with a more complex branching structure.

With the success of collapsing lakes in the Horsefly watershed, Ian Williams from the Department of Fisheries and Oceans at Nanaimo and I created a centreline river network for the Fraser River Basin, from the mouth of the river at Vancouver to the upper reaches of the Driftwood River. The river network used 1:20 000 scale TRIM data. The centreline network is the spatial structure that underlies models in the Integrated Fraser Salmon Model (IFSaM) system that simulates salmon migration in the Fraser River [94, 95].

Figure 3.23 show a subset of the Fraser River network with its connectivity properties through lakes and two-bank rivers. We selected a network edge e near the mouth of the Nechako river; the figure displays the edges upstream from e in red and the edges downstream from e in green. The river surface areas from our network for the Fraser and Nechako rivers have also been used in a model that forecasts the river temperature from the surface area and atmospheric temperature. When compared against historical data, the model with our area estimates was more accurate than the same model with older area estimates.

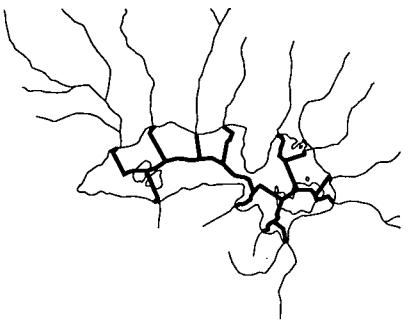


Figure 3.22: The medial axis of the lake boundary does not respect islands

A difficulty, which we have not resolved, is over-estimating the lake and river areas because we do not consider islands and sandbars. Sandbars appear along river banks and narrow the effective width of the river. Islands eliminate area from lakes. We expect that a more liberal definition of a river bank can handle most sandbars. In other cases, sandbars cross the width of rivers, which will make automatic handling more troublesome.

To negate the effects of islands on the lake areas, we can subtract the area of the islands from the area of the rivers or lakes to which the islands belong. This solution is not entirely satisfactory, since it does not give us an easy method for finding the area of a river between two points on the river banks, and the automatically generated centrelines do not respect the land formations (Figure 3.22). Although the medial axis can be computed (or approximated by our

techniques) for polygons that have holes, the resulting diagram contains loops and would require an additional decision process on how to cut the loops to form a single-line network for fisheries analysis. In our salmon migration application, lakes that were large enough to contain islands were interesting primarily for the role they played in overall connectivity; spawning occurs in smaller streams. Thus, we did not implement more general handling of islands.

Figures 3.24, 3.26, and 3.25 show additional examples of the centreline approximation. The centreline approximation in Figure 3.24 threads through a very narrow section of a lake. Figures 3.25 and 3.26 demonstrate that the approximation can provide good centrelines at different scales and for both wide and narrow rivers. These last two figures are not shown to the same scale; Figure 3.25 is a small river from the upper right corner of Figure 3.26.

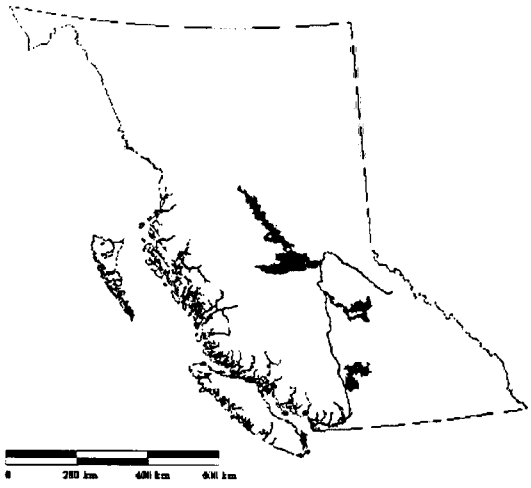


Figure 3.23: Centreline network for part of the Fraser River

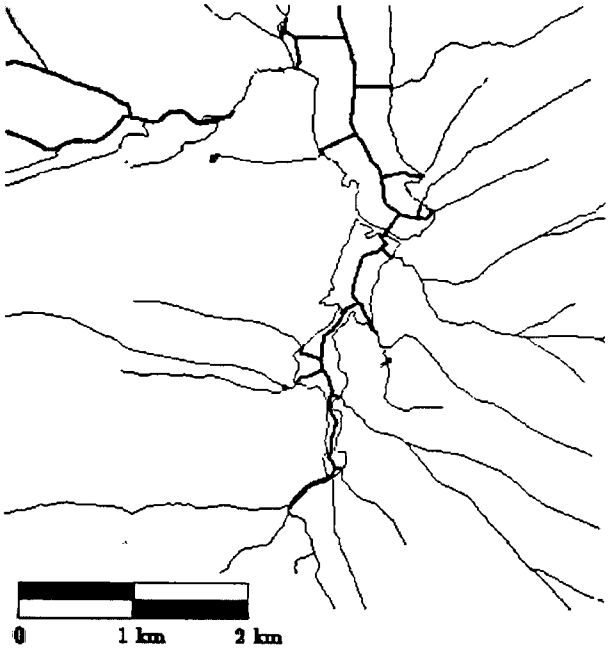


Figure 3.24: Centreline for a narrow portion of the Lower Campbell Lake

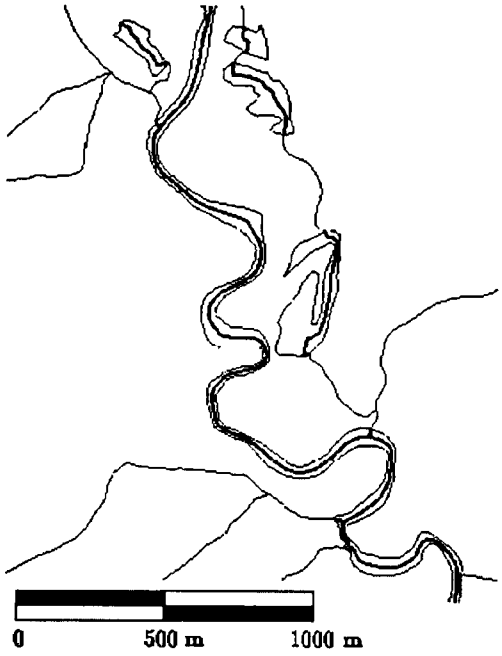


Figure 3.25: Centreline approximation for a winding section of the upper Quesnel River

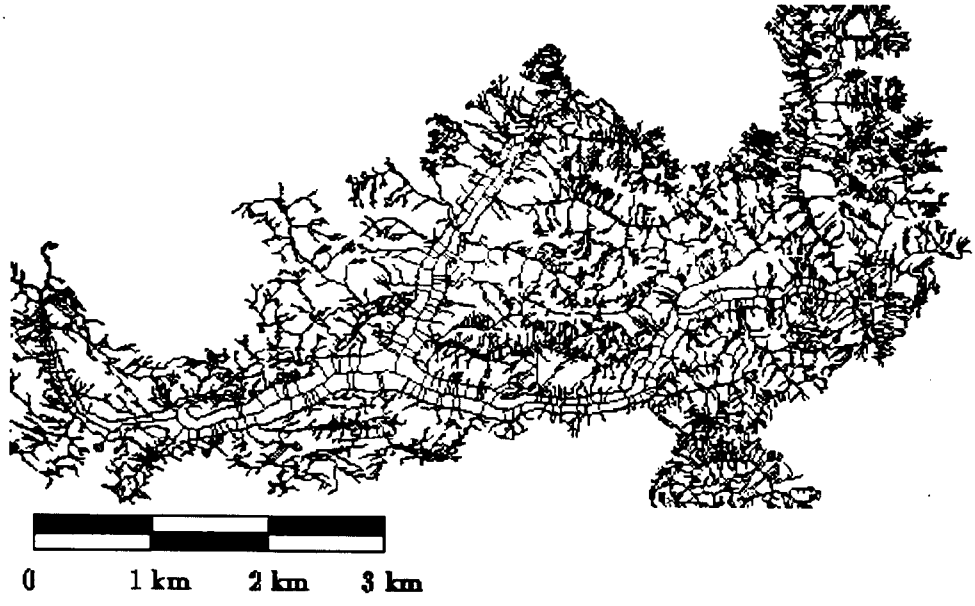


Figure 3.26: Centreline approximation for the end of the Quesnel River

Chapter 4

Terrain

In the progression from river networks to watersheds of rivers, we must model the terrain that appears inside the watersheds. This chapter summarises basic terrain representations in Section 4.1 and then focuses on data structures (Section 4.2) and algorithms (Section 4.3) for storing and creating *Triangulated Irregular Networks* (TINs).

For a broader overview of terrains and TINs, refer to Marc van Kreveld's summary article [89] and Mark Kumler's monograph [53].

4.1 Terrain Representations

There are three classic representations for terrains: contour lines, raster grids, and Triangulated Irregular Networks (TINs).

Contour lines are iso-height lines of the terrain. They first appeared in printed topographic maps and have been adopted in digital data, both from digitizing topographic maps and as a familiar representation. Visually, contour lines make it easy to locate mountains and pits because each one generates sets of nested contours. For computers, this nesting must be made explicit, whether as links between

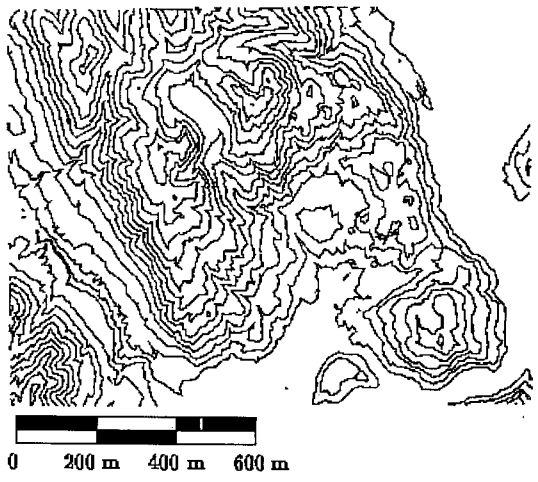


Figure 4.1: Contours of Weaver Creek, 100 metre intervals.

adjacent contour lines or through a contour tree [90].

Raster grids are more common digital models of terrain. Raster grids subdivide the terrain with a regular grid and measure the elevation at one point, such as the grid centre, within each grid cell. The raster grid has an efficient storage structure. The x and y coordinates of every point are implicit in the location of the cell within the grid. Consequently, we only store the elevation of each grid cell and the number of rows and columns in the grid. The grid also provides a topological structure for the terrain. From any grid cell, we can find the adjacent cells in constant time and every cell has a fixed number of neighbouring cells. With the compact structure and the local topology, terrain algorithms that work with raster grids are efficient. The main disadvantage of raster grids as terrain representations is the storage space. The grid must be fine enough to capture terrain features in rough regions. If there is a flat area in the same terrain, such as a lake or a plateau, then the grid also subdivides the flat area at a fine resolution since the grid is regular.

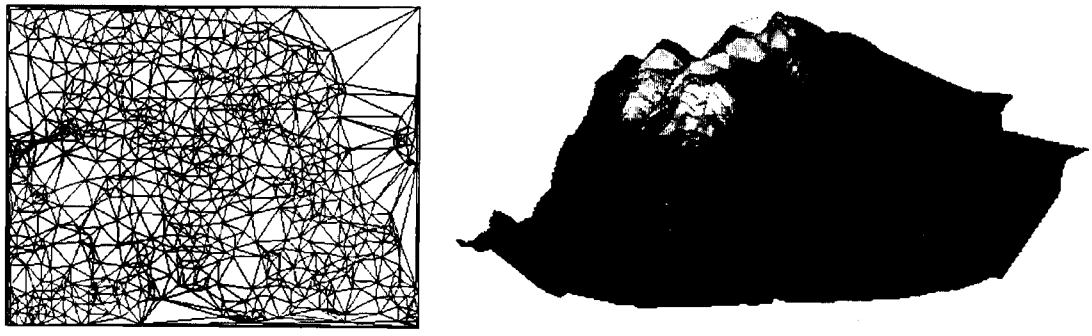


Figure 4.2: Triangulation (left) and perspective view (right) of TINs for Weaver Creek

TINs are continuous representations of terrains. They model the terrain with piecewise-planar triangular patches between selected elevation points on the terrain. Unlike raster grids, the elevation points are not necessarily at regular intervals. Consequently, TINs can represent flat regions with few sample points and use a proportionately larger number of sample points in rough landscapes. By allowing the sample points to be in irregular positions, TINs can also have higher detail around important terrain features such as valleys and ridges. The disadvantages of TINs are that every sample point must store its x, y, and elevation values and that the TIN must store face adjacency information.

There are two other terrain representations that are not frequently seen. The first is a

"richline" description of the terrain and the second uses quaternary triangular meshes (QTMs). A richline description [28] encodes the terrain ridges, valleys, pits, and peaks to represent the main characteristics. The richline description pays more attention to the drainage characteristics of the terrain and less attention to fine-grained point elevations everywhere on the terrain.

QTMs [31] are an alternate method to latitude and longitude of encoding the location of a point on the surface of the Earth. QTMs model the Earth as a triangulated approximation to a sphere and apply a numbering scheme to identify each triangular face. Each triangular face is then represented by a single point at the triangle centroid. The numbering scheme for QTMs allow us to refine individual triangles by adding more digits of precision to the number for a triangular face, which implicitly subdivides a triangle. As the QTM numbering scheme comes from a triangulated approximation to the Earth, we can approximate terrain with a set of QTM points in a manner similar to TINs. However, unlike TINs, the adjacencies between faces would be implicit in the number for each QTM point.

At present, neither the richline nor the QTM representations have had much attention for terrain analysis.

We use TINs for our terrain model since our data is not gridded and it combines lower density elevation data with river data. The following sections describe algorithms and data structures for TINs and definitions of terrain features on TINs.

4.2 TIN Data Structures

A TIN is just a planar embedded graph (a triangulation) to which we attach specific meaning: each triangular face represents part of a terrain. Consequently, we can use the adjacency lists, winged-edges, or quad-edges of Section 2.2 to represent the TIN. Also in parallel with graphs, the trade-off between structures is the time to access neighbouring edges or faces. As with graphs, the choice of representation for the TIN depends on whether our algorithms need to access information about adjacent faces or edges and whether the representation handles these accesses.

van Kreveld [89] describes another data structure available for storing TINs. It is similar to the quad-edge data structure in that it stores pointers to both the vertices and the faces of the TIN, but it is not edge-centric. The TIN is still stored as a graph, but there are graph nodes for every TIN edge, every TIN vertex, and every TIN face. Every TIN edge has a graph edge to the nodes of its end vertices and to the nodes of its adjacent faces. Similarly, every TIN face has a graph edge to the nodes of its three adjacent edges and maintains a consistent order of the edges around the triangle. The TIN vertices only act as the endpoints of TIN edges; they do not act as the start of graph edges since the degree of a vertex is arbitrarily large and we want the graph nodes to contain a fixed number of pointers to other graph nodes. This structure allows a constant-time access from any face to its adjacent faces, incident vertices, and edges. It also allows constant-time access from any edge to its immediately-adjacent edges (through the faces) and to its incident faces and vertices.

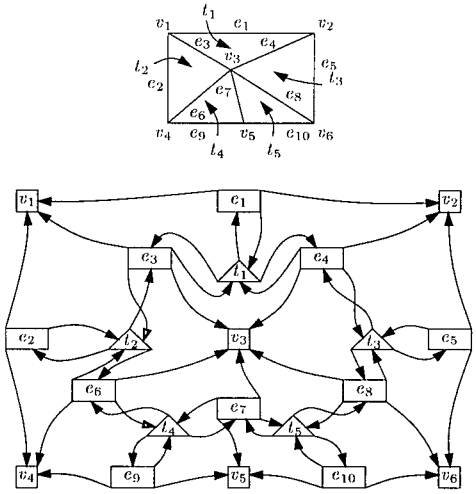


Figure 4.3: A TIN and its network structure (from [89])

Of all the data structures, we use quad-edges to store TINs since they represent both TIN vertices and faces in a symmetric manner.

4.3 TIN Algorithms

Much of the existing terrain elevation data appears as a set of contour lines, of profiles, or raster grids. There are many algorithms for converting these types of data into TINs [38, 47, 53, 89] that select a subset of the raster grid or contour points as the TIN vertices. Once we have the set of TIN vertices, a Delaunay triangulation of the vertices defines the faces of the TIN. We restrict our attention to the incremental conversion algorithm of Heckbert and Garland.

Heckbert and Garland use a user-specified error tolerance between the gridded elevation points and the TIN. They begin with a coarse TIN—a triangulation of the xy bounding box for the terrain. Their algorithm computes the vertical distance between the current TIN and every point in the raster grid. If the furthest point is within the error tolerance then the TIN creation is complete.

Otherwise, the algorithm adds the raster point with the furthest distance from the current TIN to the TIN and re-triangulates the TIN vertices to re-establish the Delaunay property. The algorithm iteratively adds points in this manner until the TIN satisfies the user's error tolerance.

Heckbert and Garland's algorithm adds points according to their vertical distance from the TIN since it is a convenient and easily-computed value. As with the polyline simplification of Section 2.4, we could apply other criteria, such as changes in terrain curvature or the change in volume below the terrain with the addition of a point [3]. However, these other criteria are often more expensive to compute after each insertion to the TIN.

Most TINs use a Delaunay triangulation to define the triangular faces since the triangulation avoids small angles in its triangles. Since the triangulation is independent of the actual terrain structure, the triangulation can introduce inconsistencies. For example, the triangulation can insert a triangle that blocks a narrow valley in the terrain (Figure 4.4). Known structural lines, such as ridges, valleys, and rivers, are often inserted into the triangulation as *breaklines* to remedy these problems [17]. The triangulation becomes a constrained Delaunay triangulation so that the breaklines are guaranteed to appear in the final triangulation. Another alternative is to manually alter the TIN so that its characteristics match those of the terrain [93].

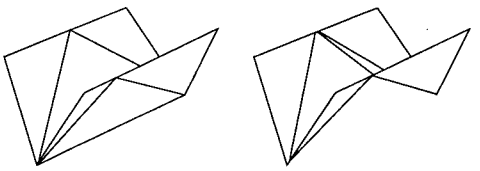


Figure 4.4: A narrow valley (left) and a bad triangulation of the valley (right)

4.4 TIN Features

The analysis of TIN drainage characteristics in Chapter 5 focuses on particular features of the TIN: ridges, valleys, peaks, and pits. Given a model of how water flows on a TIN, we use the definitions from Frank et al. [39]:

- A TIN edge e is *transfluent* if water flows from one of its adjacent triangles, across the edge, and onto the other adjacent triangle.
- A TIN edge e is a *valley* (or is *confluent*) if water from both adjacent triangles flows onto the edge.

- A TIN edge e is a *ridge* (or is *difluent*) if no water from either adjacent triangle flows onto the edge.

The definitions of pits and peaks are independent of the water flow model:

- a *peak* is a local maximum of the surface. In a TIN, a peak is a vertex with higher elevation than its incident edges and triangles.
- A *pit* is a local minimum of the surface. In a TIN, a pit is a vertex with lower elevation than its incident edges and triangles.

Chapter 5

Watersheds

The watershed of a river X is the set of terrain points that drain into river X. It is natural to ask questions about watersheds when analyzing the effects of land-based activities on the environment, on rivers, or on fisheries. Sample activities are logging, road construction, and mining.

In this chapter we investigate the properties of watershed boundaries on a TIN. While more complex models exist, we use the assumptions that underlie the majority of the complex models and show that some desirable properties of watersheds are inconsistent with even the simple model. As we expose the inconsistencies, we build a data structure called the *watershed graph* to represent the watersheds of a terrain. We prove that our data structure represents a structure that is consistent with how water flows on the TIN.

Let p be a point on a terrain T and let *trickle*(p) be the path of steepest descent on T from p. The path *trickle*(p) is the drain path of p on T. The *watershed* of point q is then all points whose drain path includes q, that is $\{p \mid q \in \textit{trickle}(p)\}$.

There are two watersheds that we might consider in a terrain. The first is the watershed of a pit in the terrain. We might want this watershed for an interest in the watershed itself, for a measure of whether or not the pit should be smoothed out of the terrain, or for a measure of how sensitive the current TIN may be to further detail refinements. The second is the watershed of an individual point on the terrain. This latter type of watershed handles the watersheds of rivers by asking for the watershed of the most-downstream point of the river.

The chapter begins with a description of previous raster and vector watershed algorithms in Section 5.1. We then outline the approach taken by our algorithm in Section 5.2. Next, Sections 5.3

and 5.4 define terrain features and assumptions used by our algorithm. Section 5.5 then presents the degeneracies that can occur in a terrain that cause inconsistencies between the derived watersheds and our expectations of the watersheds and, in parallel, develops our watershed graph to capture the watershed structure of pits on the terrain. We prove that our algorithm is correct in Section 5.6 and analyse the construction complexity for the graph in Section 5.9. Since we want to deal with data that can contain degeneracies, Section 5.7 describes some approaches for handling flat terrain. Section 5.8 describes alternative representations for watersheds. Finally, Sections 5.10 and 5.11 describe an implementation of our algorithm and lessons learned through the implementation.

5.1 Watershed Background

Most automated approaches for identifying watershed boundaries simulate rainfall and assume that water always flows along the path of steepest descent. The algorithms can be categorized as contour algorithms, raster algorithms, or vector algorithms.

5.1.1 Contour Algorithms

Manual techniques for finding watersheds often use contour algorithms. Some computer algorithms emulate these techniques. Tang [87] extracts peaks, pits, saddles, ridges, and drainage lines from contour lines by triangulating the points of the contour lines into a TIN and examining all horizontal edges of the TIN. These horizontal edges go between contours of the same elevation and represent "specific geomorphological elements" that are classified as pits, peaks, saddles, ridges, and drainage lines. Tang uses the medial axis between these contours of the same elevation to convert a set of horizontal edges or triangles into terrain features such as ridges. He concludes that this method does not find all terrain features but does identify the geomorphological elements in critical regions.

More mathematical work on watersheds characterises the shapes of watersheds. Chorley et al. [20] suggest one loop of a lemniscate of the form $\rho = \ell \cos k\theta$ in polar coordinates as a description instead of circular measures [63]. For a watershed, they use the length of the watershed as the parameter ℓ and its area to obtain the parameter k of the ideal watershed. They then classify watersheds by and comparing k values or ratios between the lemniscate perimeter and the watershed perimeter.

5.1.2 Raster Algorithms

Historically, most computer algorithms for identifying watersheds are raster algorithms. These algorithms appear in the literature for geomorphology, geology, cartography, computer vision, and civil engineering.

Collins [23] allocates one watershed per local minimum in the terrain, examines the terrain points in order of increasing elevation, and assigns a watershed value to each point that matches the watershed of an adjacent point. The computed watershed boundaries vary with the processing order of the points. The method also generates many watersheds—one per pit—that later raster methods try to avoid.

Other raster algorithms identify terrain features, such as ridges and channels, and reconstruct the watershed topology from the features. Peucker and Douglas [73] identify drainage lines by labelling the highest raster cell in every 2×2 window of the raster. The unlabelled cells are locations where water accumulates and form the drainage lines of the terrain. In this approach (and other feature detection algorithms), it is difficult to prove that the ridge networks are connected and that they correctly delimit the watersheds.

Mark [57] simulates rainfall on a raster by allocating one raindrop to each terrain point and letting the raindrops flow on the terrain; terrain points that do not receive any raindrops belong to the watershed boundaries. By recording the final destinations of each raindrop for each cell, we collect the raster cells into watersheds. Tracking the rainfall is proportional to the grid size, which can be large. Moreover, we know which cells belong to the watershed but have no indication of the boundary yet.

O'Callaghan and Mark [68] use an approach similar to Peucker and Douglas, but they adapt the algorithm to better handle noise in the data and to only delimit the major drainage lines. Their algorithm works with a 3×3 cell window, so each raster cell can be affected by its 8 neighbours. Their algorithm begins by smoothing the raster within each 3×3 window and assigning a single drainage direction for each raster cell to one of its 8 neighbours. Each raster cell is then classified by the number of incoming and outgoing drainage edges from its 8 neighbours as a pit, ridge, link, or fork. They create preliminary drainage basin labels for each tree of raster cells that flow together and use the preliminary labels to overflow pits through the lowest saddle point on their boundary; all but the lowest pits are overflowed. Then, for each raster cell c, the algorithm estimates the

watershed area by the number of raster cells that drain through c. It then identifies the major drainage lines as those cells whose area estimate exceeds a user-specified threshold.

Jensen [51] also allocates one watershed for each local minimum in the terrain, but uses a greedy steepest ascent algorithm to assign terrain points to watersheds. She uses a 3×3 window around a grid cell in her search for the steepest direction. The greedy algorithm creates problems when a watershed extends above a saddle point in the TIN. In these instances, the computed watershed can wrap around the implicit water divide at the saddle and can claim the entire cap of a mountain for the watershed when the cap actually drains to more than one location.

Douglas [28] proposes an algorithm similar to Jensen's for finding ridges and channels in a raster terrain. To find ridges, he examines each raster cell and flags the lowest corner of each cell. After examining all cells, the unflagged corners form the ridges. Similarly, to find channels, he flags the highest corner of each cell. After either of these operations, the algorithm thins the lines in the terrain (by erosion of boundary contours) until only a skeleton of the original lines remain. The ridges, channels, and other information-rich lines form a "richline model" for the terrain that captures the main drainage characteristics for the terrain. While the richline model tends to capture the watershed characteristics, ridge features may not always meet so the model may not capture the entire watershed boundary.

Band [6] uses the network edge detection of Peucker and Douglas [73] and the pit overflow technique of O'Callaghan and Mark [68] to find well-connected drainage networks and ridges in raster grids. He then considers all non-network raster cells as a drainage area and applies a line-thinning algorithm to the drainage area while preserving line connectivity across river junctions. The thinned lines are the watershed boundaries in the raster. A two-pass flood-fill algorithm then labels all the raster cells with the river name in whose watershed they appear. While Band's algorithm allows for watersheds with holes (where a separate pit occurs), the holes are not connected to the outer boundary so further processing must find whether or not one watershed is contained inside another.

Martz and De Jong [59] also model watersheds in a raster by determining the flow direction for each raster cell and collecting all the cells that drain to a common depression. For spurious depressions and flat areas, they modify the elevations in the terrain raster so that each raster cell flows towards the nearest cell with a downward slope in the original raster. Their elevation

modifications tolerate nested depressions. Later work between Martz and Garbrecht [41, 58] embeds this technique in a larger system for finding watersheds, drainage networks, sub-watersheds, and network properties such as the Strahler order.

Qian et al. [76] use an expert system to guide the connections of local drainage networks by global data. Their system begins with a low-level identification of ridges and valleys in a raster grid similar to Jensen. Unlike Jensen, their process uses a larger raster window to identify ridges and valleys (size $3N \times 3N$ for positive integers N) and restrict themselves to four possible directions for the ridges or valleys (horizontal, vertical, or either diagonal of a cell). Groups of valley cells or ridge cells are thinned into single-cell width segments and the segments are locally connected into a network according to the adjacencies of their endpoints. Connections between the low-level networks are then created with the assistance of an expert system. The expert system characterises potential connections from curvature similarities, segment lengths, elevation and slope differences, segment orientations, ridge intersections, the distance between endpoint, and a set of user-specified tolerances for these criteria. At the end of their work, they mention how to construct watersheds for their drainage networks with a tree-thickening algorithm on the network segments that maintains consistency with the ridge segments found in the lower-level processing.

Zhang et al. [98] use an underlying continuous approximation to the raster to identify watersheds. They approximate the terrain with a surface that passes through every raster point and is simplified through a discrete cosine transform. They estimate the valley and ridge raster cells from the transformed terrain with first-order and second-order partial derivatives. Next, they extend the valleys and ridges (to a limited degree) to close gaps between isolated sections. Finally, they apply a flood-fill naming scheme with unique identifiers for valley edges to label all raster cells that belong to a common watershed. As with the previous feature-detection algorithms, the extensions of valleys and ridges try to close off watersheds, but there is no control over how much extension is necessary.

Chorowicz et al. [21] only extract drainage networks from a raster grid, but their techniques could be extended to find ridges and thus provide a basis for another watershed algorithm. Their algorithm identifies well-connected drainage networks and represents areal components, such as wide valleys, in the network. Their algorithm begins with a profile scan of the raster grid. The scan examines the elevation of each raster cell in relation to its horizontal and vertical neighbours

(possibly far away when there is little or no change in elevation) and distinguishes cells as sharp valleys, wide valleys, summits, and high flat ground. The next step uses "hydrological flow modelling" to identify the drainage network. The flow modelling follows paths of steepest descent in the raster from terrain saddle points to find river channels; the channels end at depressions in the terrain. The flow modelling then overflows the depressions through their lowest saddle point to increase the connectivity of the drainage network. The result is a single-line drainage network. Finally, the algorithm overlays the wide valleys and high flat ground features of the profile scan with the network to generate a drainage network with areal components.

Fua [40] combines multiple raster views of a terrain, whether stereoscopic photos or stereo-pair photos, with grey-scale images to model 3D terrain and subsequently extract drainage networks. He uses the stereoscopic photos to estimate the curvature of the terrain in steep areas and looks for "V"-shaped terrain features as valleys. In lower-slope areas where terrain curvature is harder to estimate, grey-scale images provide additional information for tracing the paths of rivers.

In a different context, watershed algorithms are used for feature extraction in images. For example, Najman and Schmitt [65] define a hierarchical ranking of watershed edges and use the hierarchy to identify salient object boundaries in images.

There are difficulties with extracting watershed boundaries from raster algorithms. One example is that boundary edges for the watersheds are not explicitly defined. We must derive the edges and their connections to one another from the discrete raster structure.

5.1.3 Vector Algorithms

Vector algorithms generate watershed boundaries with finer resolution than raster algorithms at the expense of using non-local properties of the data.

Skea et al. [81] use the Voronoi edges of rivers edges as the watershed boundaries for the rivers. The Voronoi approach is similar to a vector equivalent to Band's line thinning algorithm for finding watershed boundaries [6]. It assumes that water is attracted to the nearest river segment. Skea et al. take the approach one step further by applying a subsequent correction to nearby terrain features. Consequently, in flat areas, the Voronoi edges between rivers act as the watershed boundaries while in more rugged areas terrain ridges act as the watershed boundaries.

Palacios-Velez and Cuevas-Renaud [70] identify valleys and ridges from a TIN by looking at

how the paths of steepest descent on the adjacent faces to an edge lead towards or away from the edge. They make the observation that the contributing area (watershed) for a river is bounded by alternating paths of steepest ascent/descent and ridges. They note that

> Clearly in order to apply this approach for the definition of contributing areas, a good routine, capable of tracking the ridge lines in ascending as well as descending direction should be developed first. However, the existence of segments of 'pure' streamlines alternating with ridge segments would further complicate the procedure.
> ([70], page 308)

but find the watersheds by dividing the TIN faces into small areas and by following the paths of steepest descent from the centre of each small area to a river or pit. Their subdivision approach converges to the watershed, but leaves itself open to potential watershed inconsistencies. However, their suggestions of embedding the steepest ascent and steepest descent paths from ridge lines into the terrain would lead to a correct algorithm. Their later work [71] considers soil types and runoff of the terrain in the watershed partitions.

Nelson, Jones, and Miller [35, 66] simplify the raster elevation data with a TIN terrain model, sub-divide triangles of the TIN according to their drainage characteristics, and group all triangles whose paths of steepest descent from their centroids end at a common local minimum in the terrain. These groups are the watersheds. The watershed boundaries are the edges of the TIN whose adjacent triangles belong to different watersheds.

Garg and Sen [42] also approximate the terrain with a TIN to find watersheds. They classify each edge of the TIN as a ridge, channel, or flow boundary segment (transfluent edge) and use paths of steepest ascent from the channel edges to identify the watershed cascades for each channel edge (the watershed cascade for an edge is the area of land that drains directly to the edge, but not from another channel edge). They model the watershed cascade for a channel edge as a band on the terrain whose edges are paths of steepest ascent from the channel endpoints. The algorithm assumes that the paths of steepest ascent only cross TIN edges until they reach a ridge or another channel edge.

Although the vector algorithms define a set of boundaries, the results are not always the complete watershed boundaries. With the Voronoi diagram of Skea, the accuracy of the boundary depends on how well the Voronoi edges match with the terrain. With the triangle assignment of Nel-

son et al., the boundary of one watershed can incorrectly degenerate to a collection of unconnected closed curves.

Wolf [96] finds watersheds as an application of surface networks [74]. He uses a surface that is twice continuously differentiable. The surface network uses the local maxima, local minima, and the saddles of the surface as the graph vertices and links the vertices by their proximity on the terrain, much like contour trees [90]. Wolf specifies a set of properties that the graph edges must satisfy to be a weighted surface network, but does not indicate how the edges should be found. The watershed of a pit p is then a closed cycle in the weighted network graph that alternates between passes and peaks, that encloses all the river courses and pits "upstream" from p, and that does not encircle any other river courses or pits. Unfortunately, the approach does not consider mathematical drainage degeneracies, such as a watershed whose interior is not connected, that can occur even in twice continuously differentiable surfaces.

5.2 Watershed Algorithm Overview

Our approach for identifying watersheds focuses on the boundaries instead of on the area inside the watersheds. As with Nelson et al., we model the terrain with a TIN and assume that water always follows the path of steepest descent. We identify a set of edges and paths on the TIN that form a superset of the watershed boundary edges for the pits in the terrain and connect these edges and paths into a graph. We prove that the graph is planar and that each face of the graph is the watershed of one pit on the terrain. Finally, a simple graph traversal algorithm extracts the watershed boundaries as a single closed and connected curve.

Most greedy algorithms for constructing TINs from raster grids [47, 53] provide good space savings over a raster terrain representation. As suggested by Fowler and Little [38] and further developed by Little and Shi [56], precomputing structural lines such as ridges and valleys of the raster grid and using these lines as a basis for a constrained triangulation of the terrain can provide an equally good terrain representation with the same error tolerance as the other TINs but use fewer TIN vertices.

We can easily question each of our model assumptions, namely that we approximate the terrain with a TIN, and that water always follows the path of steepest descent. on the terrain. For the terrain, it is non-trivial to guarantee that the drainage characteristics of the TIN and terrain

match. This aspect leads to a different research area that starts with embedding breaklines into the TIN to correct poor triangulations [17] and can lead to shaping or sculpting the terrain [93]. For the path of water, the assumption implicitly states that water has neither volume nor inertia. Once water reaches a pit, it cannot escape the pit no matter how shallow the depression in the terrain. When water reaches a sharp edge of the terrain, it can change its course by up to 90 degrees.

Despite their possible limitations, our assumptions are common for many models of surface water flow. While we limit ourselves to two simple assumptions, models with extensions of our assumptions contain the properties of our model. Our goal is to obtain *consistent* watershed boundaries for our simple model; once we have consistent boundaries we can better isolate how additional properties affect the watersheds of points.

5.3 Height Profile Function

The drainage characteristics at a point depend on the elevations of the surrounding TIN. Define the *height profile function* $h_{\epsilon,p}(\theta) : [0, 2\pi) \rightarrow \mathbb{R}$ at a point p to be a function of the angle θ that returns the elevation at which the cylinder $\{(x - p.x)^2 + (y - p.y)^2 = \epsilon^2,\ z \in \mathbb{R}\}$ intersects the TIN in a direction θ from p. This definition can be seen as a continuous version of the 8-neighbor test of O'Callaghan and Mark [68]. We will only talk about the profile around one point, which we assume to be the origin of the coordinate system. The interesting properties of $h_{\epsilon,p}(\theta)$ are independent of ϵ for small ϵ (Lemma 5.3.1) so we denote the height profile function as $h(\theta)$.

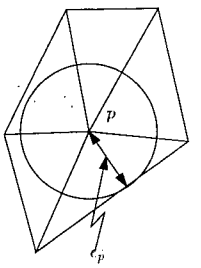


Figure 5.1: The maximum radius for the height profile function around p

Lemma 5.3.1 *The maxima, minima, and sign of the height profile function* $h_{\epsilon,p}(\theta)$ *on a TIN around a point* p *are independent of* ϵ *for sufficiently small* ϵ.

Proof: Let ϵ_p be the distance from p to the closest vertex of the TIN other than p and to the closest edge in the TIN not adjacent to p (Figure 5.1). Since the TIN is piecewise planar around p, for any ϵ_1, ϵ_2 that satisfy $0 < \epsilon_1 \leq \epsilon_2 < \epsilon_p$, the height profile functions are related to one another: $h_{\epsilon_1,p}(\theta) = \frac{\epsilon_1}{\epsilon_2} h_{\epsilon_2,p}(\theta)$. Since $h_{\epsilon_2,p}(\theta)$ is a positive fraction of $h_{\epsilon_1,p}(\theta)$ they share the same local maxima, local minima, and sign. ∎

In an implementation that evaluates the sign and critical points of $h(\theta)$, an equivalent definition for the height profile uses an ϵ-sphere centred at p to intersect the TIN instead of an ϵ-cylinder. The two functions have the same local maxima and local minima as well as the same sign (Lemma 5.3.2). The main difference between the two definitions is whether a vector on the TIN has its length normalized by its full length $\sqrt{x^2+y^2+z^2}$ or by the length of its projection onto the xy-plane $\sqrt{x^2+y^2}$.

Lemma 5.3.2 *The height profiles of a unit cylinder and a unit sphere when cut by a plane, all centred at the origin, have the same sign and the same local minima and maxima.*

Proof: Let z_{cyl} be the height profile from a unit cylinder centred on the origin and a plane through the origin. For a given value of θ, the intersection point on the unit sphere is along the vector from the origin to the intersection point on the cylinder, so the height profile z_{sph} from a unit sphere centred on the origin is

$$z_{sph}(\theta) = \frac{z_{cyl}(\theta)}{\sqrt{1+z_{cyl}^2(\theta)}}$$

Since the normalization factor is strictly positive, both $z_{cyl}(\theta)$ and $z_{sph}(\theta)$ have the same sign for all values of θ.

Next, we show that the two profile functions have the same local maxima and minima by showing that their derivatives have the same zero values.

Differentiating with respect to θ yields

$$\begin{aligned}
z'_{sph}(\theta) &= \frac{z'_{cyl}(\theta)}{(1+z_{cyl}^2(\theta))^{\frac{1}{2}}} - \frac{2z_{cyl}^2(\theta)z'_{cyl}(\theta)}{2(1+z_{cyl}^2(\theta))^{\frac{3}{2}}} \\
&= \frac{z'_{cyl}(\theta)}{(1+z_{cyl}^2(\theta))^{\frac{1}{2}}}\left(1 - \frac{z_{cyl}^2(\theta)}{1+z_{cyl}^2(\theta)}\right) \\
&= \frac{z'_{cyl}(\theta)}{(1+z_{cyl}^2(\theta))^{\frac{1}{2}}}\left(\frac{1+z_{cyl}^2(\theta)-z_{cyl}^2(\theta)}{1+z_{cyl}^2(\theta)}\right) \\
&= \frac{z'_{cyl}(\theta)}{(1+z_{cyl}^2(\theta))^{\frac{3}{2}}}
\end{aligned}$$

Since the denominator is always greater than 0, the derivative $z'_{sph}(\theta)$ is zero exactly when $z'_{cyl}(\theta)$ is zero so the two functions have their extrema at the same locations. The correspondence between maxima and minima for the profile functions follows from this result. ∎

For a plane $ax + by + cz = 0$, the height profile is $h(\theta) = \frac{\epsilon}{c}(-a\cos(\theta) - b\sin(\theta))$. In the domain $[0, 2\pi)$, this height profile is differentiable, has one maximum point, and has one minimum point; the extrema occur where $\tan(\theta) = \frac{b}{a}$.

When multiple planes meet at the point p, such as at a vertex in a TIN, the angle range $[0, 2\pi)$ is partitioned into intervals, one for each plane. Within each of these intervals, the height profile is differentiable. At the interval boundaries, the height function is continuous. Each angle interval can host either one local maximum and one local minimum at the interval boundaries or one local maximum and/or local minimum inside its interval. If there are n planes meeting at a point then the height profile for that point can have at most $\frac{n}{2}$ local maxima and $\frac{n}{2}$ local minima, with equality attained when these extreme values are on edges where the planes meet.

The local maxima and local minima of the height profile for a point p correspond to ridges and valleys at p. A local maximum identifies a direction where the flow of water splits around p: a ridge. A local minimum identifies a direction where the flow of water collects at p: a valley.

The values of the height profile at the local maxima and the local minima are either positive, negative, or zero. A local maximum or local minimum with height 0 occurs when there is a horizontal edge or face adjacent to the point. For this section, we assume that these horizontal sections of the TIN already have flow directions assigned to them that allow us to perturb the extreme point so that they either have a positive or a negative sign; the remainder of this section assumes that there are no horizontal edges in the TIN.

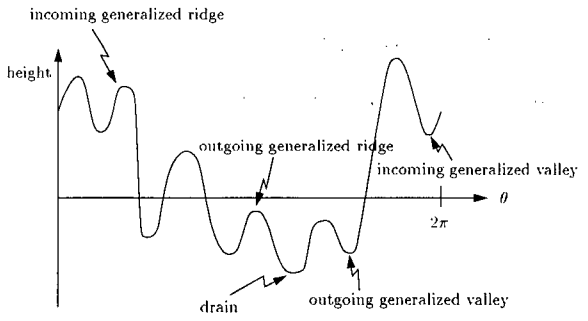


Figure 5.2: A sample height profile

Name the angles for the local maxima and local minima as follows:

- an *incoming generalized ridge* is a local maximum whose height profile value is positive,
- an *outgoing generalized ridge* is a local maximum whose height profile value is negative,
- an *incoming generalized valley* is a local minimum whose height profile value is positive, and
- an *outgoing generalized valley* is a local minimum whose height profile value is negative.

Figure 5.2 shows examples of these angles against a sample height profile function. The names include the word *generalized* to emphasize that the maxima and minima are not necessarily TIN edges even though they behave like ridges and valleys. When the path of steepest ascent out of a point p is along a TIN face incident to p then that path corresponds to a local maximum of the height profile function for p. Since the path isn't an actual TIN edge, we call it a generalized ridge. A similar situation occurs when a path of steepest ascent into p reaches p from across a TIN face to produce an outgoing generalized valley at p.

If the height profile value is negative at the global minimum then the direction of the global minimum is *drain* for the point (Figure 5.2). We assume that individual TIN points have at most one drain. Otherwise, the height profile value at the global minimum is positive and the point is a pit of the TIN.

In a height profile, as in all continuous functions, the local maxima alternate with the local minima. The alternations between the terrain features at the extrema follow a pattern of their own. An outgoing generalized valley can immediately precede or follow any kind of local maximum. An incoming generalized valley can only precede or follow a incoming generalized ridge. An incoming generalized ridge can precede or follow any kind of local minimum. Finally, an outgoing generalized ridge can only precede or follow an outgoing generalized valley.

Label the local minima and maxima of the height profile function by $\{m_1, m_2, \ldots, m_k\}$ where m_1 is a local minimum (wrapping the domain at $\theta = 2\pi$ back to $\theta = 0$). Then m_i is a local maximum for even i and a local minimum for odd i. For odd i, we say that the generalized ridges m_{i-1} and m_{i+1} bound the generalized valley m_i, just as the corresponding ridges in a terrain would bound a valley.

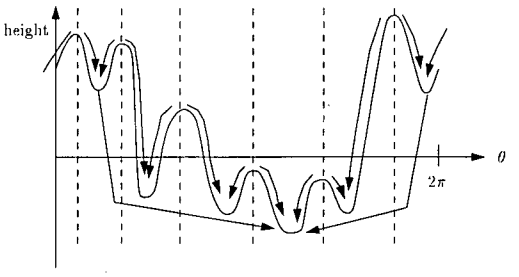


Figure 5.3: Water flow in a height profile

The values of θ that correspond to generalized ridges partition the θ-axis into intervals, each of which bounds a generalized valley (when $\theta = 2\pi$ is wrapped to $\theta = 0$). The generalized ridges divide waterflows: any water that appears within an interval I flows towards the generalized valley inside the interval I. Water in incoming generalized valleys flows to p and, in turn, flow along the drain of p. Outgoing generalized valleys are lower than the point p and flow away from p. So,

the incoming generalized ridges around a point p bound the region in which water flows into the point p while the outgoing generalized ridges bound the region in which water does not reach p (Figure 5.3).

5.4 Water Flow Assumption

Throughout our analysis, we assume that water always follows the path of steepest descent on a terrain. This assumption simplifies the analysis of drainage characteristics. In particular, it gives an alternate characterization of the local points around a point p on a plane π that drain through p: the points in an ϵ neighbourhood of p on π that drain through p lie on the path of steepest ascent from p on π. When our terrain involves more than one plane, we can concentrate on differences in flow at TIN edges and vertices.

One difference between paths of steepest ascent and paths of steepest descent on a TIN is that, under non-degenerate conditions, a point p has a unique path of steepest ascent but can belong to paths of steepest descent that enter p from different directions. For example, a point in a valley will belong to a path of steepest descent from either slope of the valley and will drain the valley edge. Every generalized ridge of the height profile function for p indicates a direction around p where p belongs to a path of steepest descent. Consequently, we will talk about the paths of steepest ascent out of p in the plural to refer to the paths of locally steepest ascent at p. These paths then capture all the paths of steepest descent that flow directly through p.

When a point has an incoming generalized valley, it drains an area of an ϵ-sector around the point. This area is bounded by paths of steepest ascent. We use this bounding characteristic to define the watersheds of points without having to use every single point of the TIN.

5.5 Drainage Characteristics

We expect that the watershed for a pit or the mouth of a river R is an irregular polygon that contains R, whose interior is a connected set, and whose boundary does not intersect itself. This section demonstrates that these idealized watersheds occur only in a subset of all possible TINs under our water flow assumption.

We characterise sets of terrains by the types of degeneracies that occur in the drainage

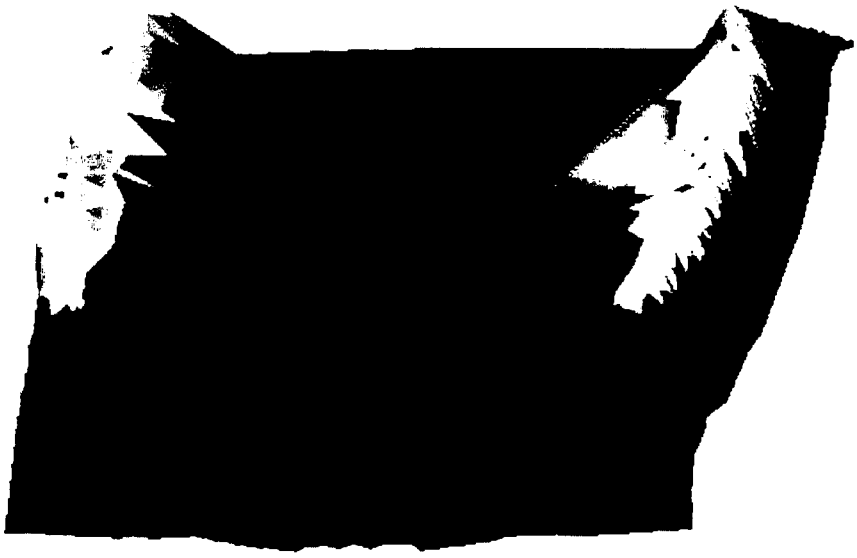
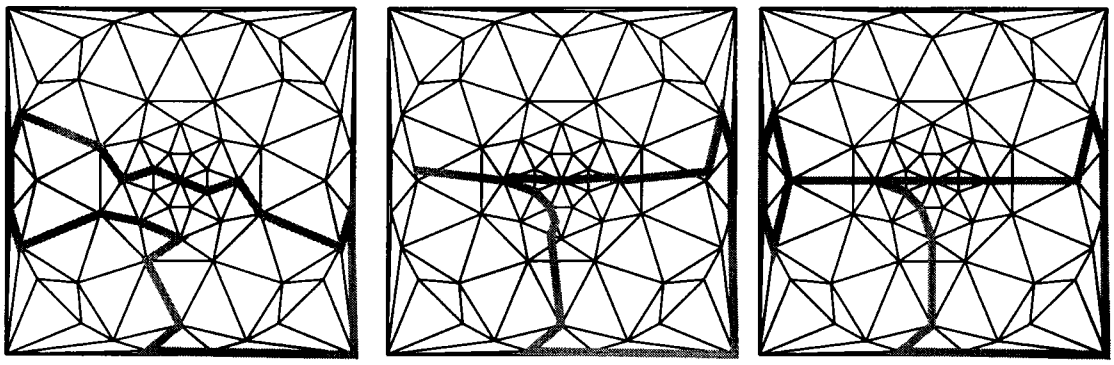


Figure 5.4: Perspective view of a sample terrain

network and label the terrains as nice, normal, or nasty. In a *nice terrain*, the local maxima and minima for the height profile functions of every TIN point only occur at TIN ridges and valleys; this is the most restricted subset of TINs. The watersheds on a nice terrain meet our expectation of watersheds. In a *normal terrain*, the local maxima and minima for height profile functions of TIN points can occur on faces of the TIN, but the paths of steepest ascent in the direction of local maxima for the height profile function must end in the interior of ridges. We expect most TINs generated from Delaunay triangulations of elevation points to be normal terrains. The boundary of watersheds on normal terrains can include degeneracies in the form of dangling polygonal lines. In a *nasty terrain*, there are no restrictions on the drainage characteristics of the terrain. Not only can the boundary of a watershed on a nasty terrain be degenerate, but the interior of the watershed itself can be several disjoint open sets. Figure 5.4 shows a terrain with two mountains that qualifies as a nice, normal, or nasty terrain depending on how we change the elevations of 8 TIN vertices. One watershed for each type of terrain appears in Figure 5.5.

We define a watershed graph for each of these terrain types. The watershed graph abstracts the watershed boundaries on the terrain and always has a planar embedding from their relation to the TIN. Each face of the graph corresponds to a single watershed in the TIN.

Section 5.5.1 describes the watershed graph and relates it to nice terrains. Sections 5.5.2 and 5.5.3 add degeneracies to the terrain and track the effects of these changes on the watershed



nice terrain normal terrain nasty terrain

Figure 5.5: Possible watersheds on terrains

graph.

5.5.1 Nice Terrain

In a *nice terrain*, every TIN vertex has the local maximum and local minimum of its height profile function along a ridge edge or a valley edge of the TIN. Consequently, the set of ridge edges in the TIN form an embedded planar graph where each face either contains a pit or drains through one corner of the face. We want to capture this planar graph as our watershed graph and modify its edge adjacencies to merge the faces that have no pits with the faces into which they drain. In this section, we will explore the drainage characteristics on the terrain, describe the types of watersheds that will occur, and then formally define the watershed graph.

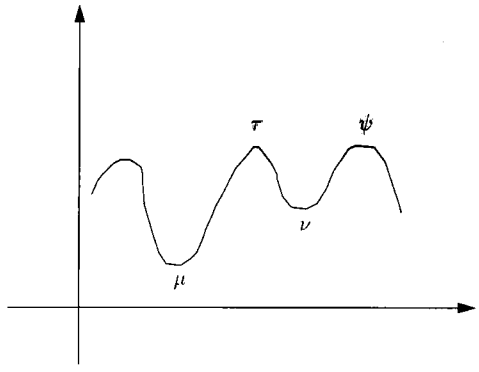


Figure 5.6: Bounding ridges for a valley

Let angle μ be the drain of p. Let ν be a local minimum of the height profile bounded by ridges τ and ψ (Figure 5.6). For the outgoing valley $\nu \neq \mu$, the ridges τ and ψ, whether incoming or outgoing ridges, prevent any water in the valley ν from reaching the drain μ. Although the water that collects in ν may join the water that collects in μ later, the ridges prevent the valleys from merging at p.

When $\nu = \mu$, we are looking at the drain of p. The water from every incoming valley at p

flows into μ; the point p is the confluence of these valleys.

Watersheds from nice terrains have two desirable characteristics: the interior of the watershed W is a connected set and every boundary point of the watershed belongs to the closure of the interior of W. These watershed characteristics correspond to our intuitive notion of a watershed and are typical of the results for watershed extraction algorithms in the literature. However, the claims in the literature are made for normal terrains. for normal terrains.

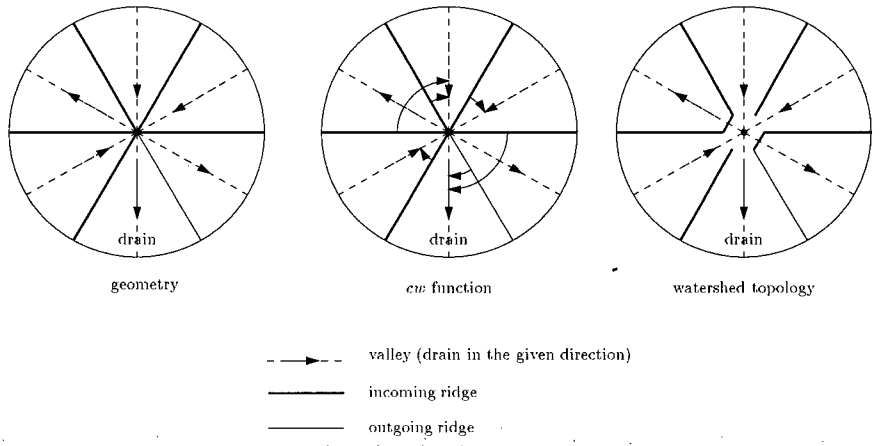


Figure 5.7: Watershed graph topology on a nice terrain.

We abstract the drainage characteristics as a *watershed graph*. We start with definitions about TIN features that we use to define the watershed graph:

- S is the set of TIN vertices whose height profile functions have more than one outgoing valley. Let p be a point of S.
- R is the set of ridges in the TIN and $R_p \subseteq R$ is the set of ridges at p (whether incoming or outgoing).
- u_{p0} is the drain of p.
- $U_p = \{u_{p_1}, u_{p_2}, \ldots, u_{p_k}\}$ is the set of incoming valleys of p sorted in counterclockwise order around p when starting at the drain u_{p0}.
- function $cw_p(r) : R_p \rightarrow U_p \cup \{u_{p0}\}$ is the drain or the incoming valley that is immediately clockwise from ridge r around the point p.

From on these definitions, we define an embedded graph $G = (V, E)$, called *the watershed graph*,

with vertices

$$V = R \cup \bigcup_{p \in S} (U_p \cup \{u_{p0}\})$$

and edges

$$E = \bigcup_{p \in S} \bigcup_{r \in R_p} (r, cw_p(r))$$

that abstracts the local drainage characteristics of the TIN (Lemma 5.5.1). For the embedding, the order of the edges around a vertex in the watershed graph is the same as the geometric order of the ridges around the corresponding point in the TIN. The order of the edges determines the faces of the watershed graph and the existence of a planar embedding (Lemma 5.5.2). The watershed graph for nice terrains is bipartite, but this property will not hold once we extend the graph definition for nasty terrains.

Figure 5.8 shows a triangulation of a terrain (left) with the ridges as bold lines and the corresponding embedding of its watershed graph (right).

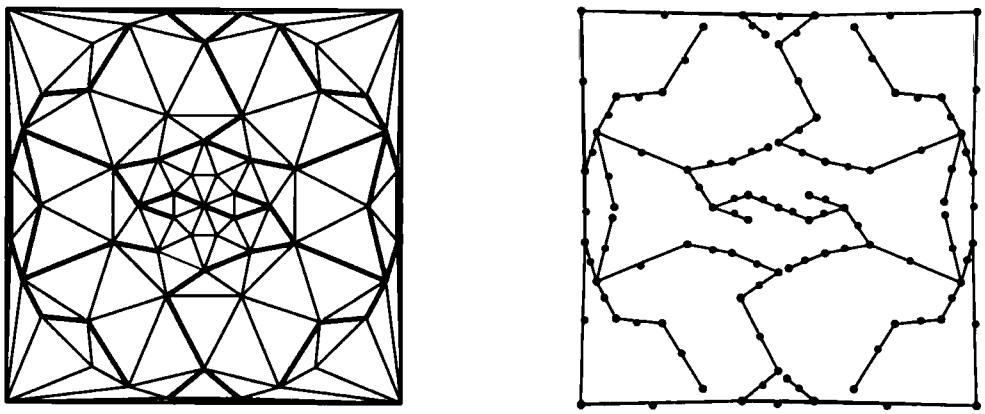


Figure 5.8: A nice terrain (left) and its watershed graph (right). Ridges appear in red.

Lemma 5.5.1 *Let q be a point on a nice TIN in a local neighbourhood N of TIN vertex p. There is a path in the embedded watershed graph between p and q that stays in N iff q drains to p or p drains to q.*

Proof: Let ν be a valley of the TIN around a point p and let μ be the drain of p. Suppose that ν is an outgoing valley bounded by ridges τ (clockwise of ν) and ψ (counterclockwise of ν).

If ν is not the drain μ then $cw_p(\tau) = cw_p(\psi)$ since τ and ψ are only separated by ν and the face of the watershed graph that contains ν is closed at p. If ν is the drain μ and there is an incoming valley at p then $cw_p(\tau) \neq cw_p(\psi)$ and the face of the watershed graph that contains ν is open to anything that reaches the point p.

Otherwise, ν is an incoming valley then $cw_p(\psi) = \nu$ and $cw_p(\tau)$ is the clockwise predecessor of ν around p; the face that contains ν is not closed at p between τ and ψ so water along ν will reach p and hence μ. Finally, if ν is the drain μ and there is no incoming valley at p and p is a peak then all cw_p values at p are u_{p0} and all faces in the watershed graph are closed at p—no water runs around or over this peak. ∎

Lemma 5.5.2 *The watershed graph has a plane embedding.*

Proof: Around each TIN vertex p, clip all TIN edges incident to p by a 2ϵ-ball around p. For each watershed graph vertex v, if v comes from a ridge r then embed v at the midpoint of r in the TIN. Otherwise, v comes from a valley u_{pi}. Embed v at the intersection of u_{pi} and an ϵ-ball centred at p. Embed the edges of the watershed graph as straight lines between their endpoints. ∎

5.5.2 Normal Terrain

Normal terrains relax an assumption of nice terrains: we allow the local minima and maxima of the height profile for a particular point to occur along a face of the TIN. In its place, we assume that all paths of steepest ascent from any TIN vertex end at the interior of a ridge (Figure 5.9), which was implied for nice terrains.

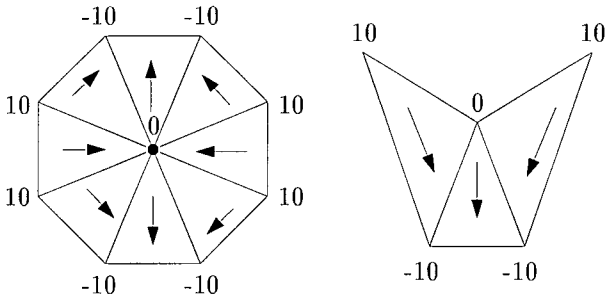


Figure 5.9: Drain along a face through a point: plan view with elevations (left) and side view (right)

Let τ be an incoming generalized ridge from a point p that is along a face of the TIN. The incoming generalized ridge τ serves two functions: as a ridge that separates the waterflows to the right of p from the waterflows to the left of p and as a valley where the water that lies precisely along τ flows into p. In this subsection, we will

define the watershed graph for normal terrains without making any changes to the TIN. However, we will begin by describing how the TIN could be changed so that we could apply the results of Section 5.5.1. We will actually apply these changes to the watershed graph instead and leave the TIN unmodified.

For an incoming generalized ridge, we can modify the TIN to reflect the multiple roles of τ and then apply the analysis of Section 5.5.1. Take a path of steepest ascent t from p in the direction of τ. Replicate t five times, separate the copies by ϵ, and number the copies t_1 through t_5 counterclockwise around p. Elevate t_2 and t_4 by ϵ and re-triangulate the TIN with t_1 through t_5 as TIN edges. Paths t_2 and t_4 are ridges in the new TIN (Figure 5.10). The new TIN has the same water drainage structure in the direction of τ from p except that the ridge and valley behaviours of τ are embedded into the terrain. There are now three regions around p; the region between t_2 and t_4 that acts as a valley into p, the region clockwise of t_2, and the region counterclockwise of t_4. The last two regions flow to separate sides of p. As described in Section 5.5.1, water between ridges t_2 and t_4 must be allowed to drain into the point p.

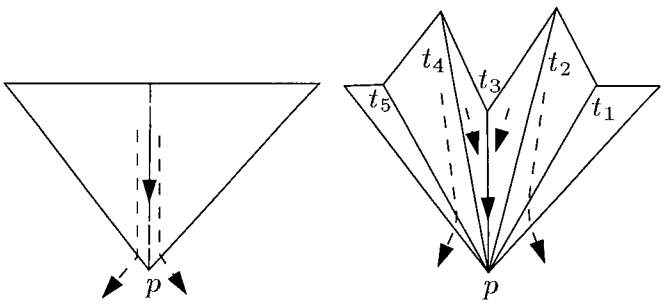


Figure 5.10: A ridge along a face and a modification to the TIN to achieve the same effect

If τ is an outgoing generalized ridge from p then τ can only bound outgoing generalized valleys. Unlike incoming generalized ridges, outgoing generalized ridges along a face only play one role for the point p: they act as a ridge to divide the waterflows among the two adjacent local minima. As with the incoming generalized ridges, we can modify the TIN so that τ appears as a ridge in the TIN: replicate the path τ three times, separate the copies by ϵ and elevate the middle copy by ϵ to become a ridge. Once again, the ridge τ prevents the water in the bounded outgoing valleys from merging with one another at p.

We can now consider how these changes affect the watershed graph of Section 5.5.1. The generalized valleys dictate the edge endpoints in the watershed graph so there is no difference between a generalized valley along a valley edge or along a face. When defining the graph topology, we only use knowledge of whether the valley is incoming or outgoing.

The incoming generalized ridges along faces in the TIN change the characteristics of the watersheds in the TIN. As with nice terrains, the interior of a watershed is still a single connected set, but the pseudo-valley along an incoming generalized ridge produces a "spike" that radiates out of the watershed. A spike has no enclosed area, so not every boundary point of the watershed is adjacent to interior points of the watershed.

The only difference between nice and normal terrains is that generalized incoming ridges or valleys can lie on TIN faces. We need to reflect these differences in the watershed graph that was started in Section 5.5.1. Since valleys do not define watershed boundaries, only generalized ridges on faces change the watershed boundary relative to nice terrains. There are two key differences in the definition of the watershed graph:

- incoming generalized ridges along faces have the effect of two parallel ridges, and
- incoming generalized ridges along faces are also treated as one incoming generalized valley.

We extend the notation from the previous section as follows:

- S is the set of TIN points whose height profile functions have more than one outgoing generalized valley. Let p be a point of S.
- R is the set of ridges in the TIN and $R_p \subseteq R$ is the set of ridges incident to the point p
- u_{p0} is the drain at p
- V_p is the set of incoming generalized valleys at p. This is the same as U_p for nice terrains.
- $A_p = \{a_{p_1}, a_{p_2}, \ldots, a_{p_l}\}$ is the set of incoming generalized ridges along faces at p in counterclockwise order around p starting at u_{p0}
- $U_p = V_p \cup A_p = \{u_{p_1}, u_{p_2}, \ldots, u_{p_k}\}$. The elements of the set are in counterclockwise order around p starting at u_{p0}
- $T_p = \{t_{p_1}, t_{p_2}, \ldots, t_{p_{2l}}\}$ are paths of steepest ascent from p that stop as soon as they encounter an element of R. Path t_i is the path of steepest ascent along incoming generalized ridge $a_{\lfloor i/2 \rfloor}$. Let b_i be the element of R where the highest point of t_i ends.
- $cw_p(r) : R_p \cup T_p \rightarrow U_p \cup \{u_{p0}\}$ is defined as follows:
  - if $r \in R_p$ then $cw_p(r)$ is the first element of $U_p \cup \{u_{p0}\}$ immediately clockwise from ridge r at p

- if $r \in T_p$ as $r = t_{p(2i+1)}$ for an integer i then $cw_p(r)$ is the first element of $U_p \cup \{u_{p0}\}$ immediately clockwise from r at p
- otherwise, $r \in T_p$ as $r = t_{p(2i)}$ for an integer i so define $cw_p(r)$ as the element of U_p that matches a_i.

Then, the embedded watershed graph $G = (V, E)$ for a normal terrain has vertices

$$V = R \cup \bigcup_{p \in S} (U_p \cup \{u_{p0}\})$$

and edges

$$E = \bigcup_{p \in S} \left(\bigcup_{r \in R_p} (cw_p(r), r) \cup \bigcup_{t_i \in T_p} (cw_p(t_i), b_i) \right)$$

and abstracts the local drainage characteristics of normal terrains (Lemma 5.5.3). For notational convenience, the definition for the watershed graph edges assumes that we can derive the ridge b_i, which is where the path of steepest ascent t_i ends, from t_i. The order of the edges around a

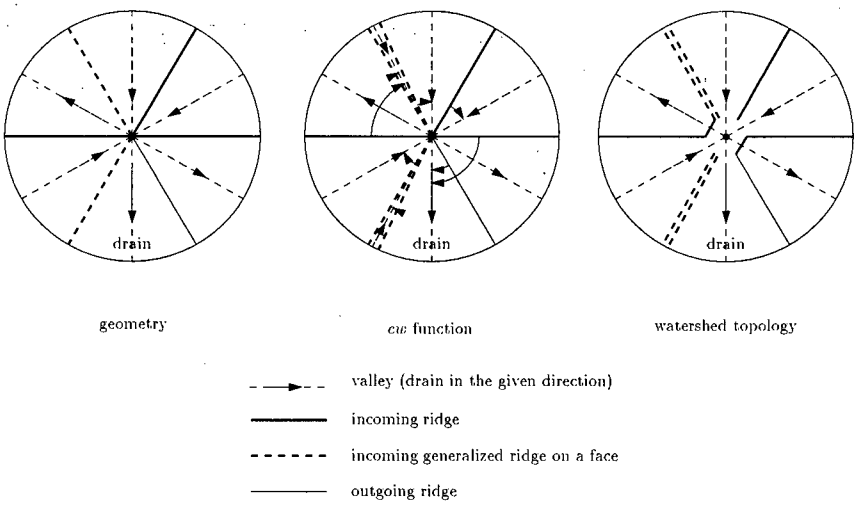


Figure 5.11: Watershed graph topology on a normal terrain.

vertex in the watershed graph embedding is the same as the geometric order of the ridges around the point in the TIN. For two paths of steepest ascent $t_{p,2i+1}$ and $t_{p,2i+2}$ with the same geometric coordinates, the edge for $t_{p,2i+1}$ appears counterclockwise from edge $t_{p,2i+2}$ at their common upper point b_i; by definition of the cw_p function, they have different endpoints at their lower endpoint. The watershed graph embedding remains planar (Lemma 5.5.4).

Figure 5.12 depicts a triangulation for a terrain (left) with ridges in bold lines and paths of steepest ascent across faces in dashed lines. The corresponding watershed graph appears in the right panel of the same figure.

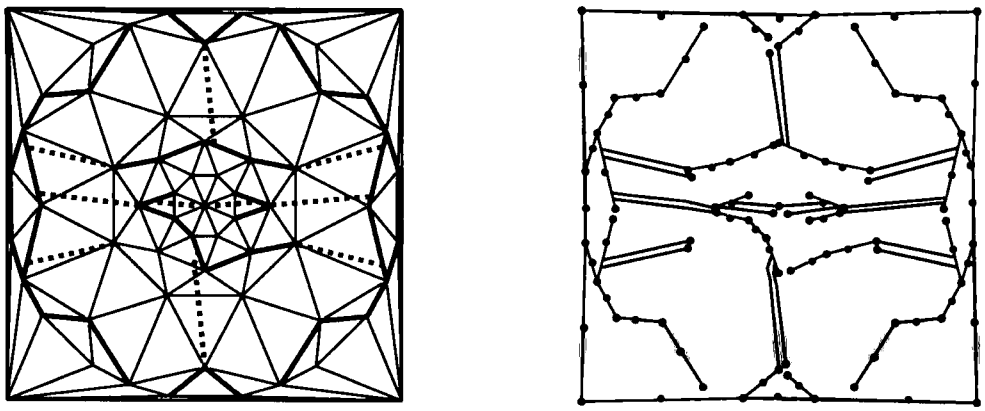


Figure 5.12: A normal terrain (left) and its watershed graph (right).
Ridges appear in red and steepest ascent paths appear as dashed lines.

Lemma 5.5.3 *Let q be a point on a normal TIN in a local neighbourhood N of TIN vertex p. There is a path in the embedded watershed graph between p and q that stays in N iff q drains to p or p drains to q.*

Proof: We show that the watershed graph captures the drainage characteristics of generalized ridges that lie on TIN faces. The analysis for the drainage characteristics of ridges along TIN edges is the same as in the proof of Lemma 5.5.1.

Earlier, we described how an incoming generalized ridge was modelled by two ridges that bound an incoming valley. These two ridges correspond to matching steepest ascent paths $t_{p,2i+1}$ and $t_{p,2i+2}$ in the watershed graph. Since the paths have different lower endpoints, the incoming valley bounded by the paths flows into the point p.

For an outgoing generalized ridge across a face, the watershed graph has no corresponding edge. Lemma 5.6.9 proves that the outgoing generalized ridges have no effect on the watershed boundary for normal TINs so excluding them from the watershed graph is acceptable. ∎

Lemma 5.5.4 *The watershed graph for normal terrains is planar.*

Proof: The embeddings for vertices of U_p and R remain unchanged from Lemma 5.5.2 of Section 5.5.1 on nice terrains. Graph edges that correspond to ridges of R_p are straight-line embeddings along the ridge edges. Graph edges from T_p, which are paths of steepest ascent along faces, follow the paths of steepest ascent in the TIN. The paths of steepest ascent guarantee that edges from different paths of steepest ascent will not cross one another. This leave us to consider the embedding of edges $t_{p,2i+1}$ and $t_{p,2i+2}$ that originate from a common path of steepest ascent a_i. Convolve the path a_i with a disk of radius ϵ and embed $t_{p,2i+1}$ and $t_{p,2i+2}$ along the outer edges of this convolution to guarantee a 2ϵ separation between $t_{p,2i+1}$ and $t_{p,2i+2}$. ■

5.5.3 Nasty Terrain

Nasty terrains have no assumptions on where paths of steepest ascent stop. Nasty terrains allow paths of steepest ascent to end at saddle points or endpoints of ridges. This situation requires a precarious alignment of TIN vertices. Nevertheless, the water flow and terrain model descriptions do not forbid this behaviour.

The only place where paths of steepest ascent affect the drainage characteristics is at incoming generalized ridges along TIN faces. The paths of steepest ascent that stop at ridges create "spikes" in the watershed boundary and were studied in Section 5.5.2. When the upper end of a spike is a TIN vertex v, all the water that drains to v can drain along the spike. If the watershed for v has a non-zero area then the spike connects two areas into one watershed: it funnels one area into another. In this instance, the interior of the watershed can be disconnected.

Let S be the set of TIN points with more than one outgoing generalized valley. Add the TIN points that are the lower endpoints of ridges to S. We have two cases to examine for steepest ascent paths that end at a point of S:

- the upper end of a steepest ascent path reaches the point of S through the drain, and
- the upper end of a steepest ascent path reaches the point of S through an outgoing generalized valley that is not the drain.

Consider what happens to a pair of parallel ridges separated by an ϵ-width valley that reach a point p of S. Since the path of steepest ascent reaches the point itself, it must approach the point

along an outgoing generalized valley. The difference is whether or not the outgoing generalized valley is the drain for the point. Start with the case where the outgoing generalized valley is the drain of p. Any water that reaches an incoming generalized valley at p must flow through p and out the drain for p. When a path of steepest ascent is converted into a pair of parallel ridges, all water that would normally follow the path of steepest descent is modelled as flowing down the valley between the two parallel ridges. Conversely, if the outgoing generalized valley is not the drain of p then any water in incoming generalized valleys must not enter the valley between the two parallel ridges.

When the paths of steepest ascent end at vertices, the two parallel paths that model the steepest ascent path form a funnel; water on the terrain can reach into the valley between the parallel paths. These ϵ-width funnels connect disjoint areas of the terrain. Consequently, the watersheds for nasty terrains can have disconnected interiors as well as spikes.

The effects of the funnels on the watershed graph are in the endpoints of graph edges. In normal terrains, the endpoints of graph edges are always from a subset of the graph vertices, namely R, whose elements were in a one-to-one correspondence with TIN features. In nasty terrains, the graph edges can stop at elements of U_p. With this change, the watershed graph is no longer bipartite.

We present the definitions for sets of the TIN that define the watershed graph. Only those definitions that change from the normal terrains are described below. We consider ridges in the TIN as open edges; they do not contain their endpoints.

- S is the set of TIN points whose height profile function have more than one outgoing generalized valley or that are the lower endpoint of a ridge. Let p be a point of S.
- The set of steepest ascent paths T_p is defined as for normal terrains. However, the point b_i where a steepest ascent path t_i ends is either the interior point of a ridge in R or is a point of S.
- I_p is the set of paths in T_q that, for any $q \in S$, end at p.
- the domain of the function $cw_p(r)$ changes to include I_p.
  $cw_p(r) : R_p \cup T_p \cup I_p \rightarrow U_p \cup \{u_{p0}\}$ is as follows:
  - if $r \in R_p$ or $r \in I_p$ and r does not enter p along its drain then $cw_p(r)$ is the first element

of $U_p \cup \{u_{p0}\}$ immediately clockwise from ridge r at p

- if $r \in T_p$ as $r = t_{pi}$ and i is odd then $cw_p(r)$ is the first element of $U_p \cup \{u_{p0}\}$ immediately clockwise from r at p
- if $r \in T_p$ as $r = t_{pi}$ and i is even then $cw_p(r)$ is the element of U_p that matches $a_{\lfloor \frac{i}{2} \rfloor}$.
- if $r \in I_p$ enters p along its drain, $r = t_{qi}$ for some $q \in S$ and i is odd then $cw_p(r) = u_{p0}$
- if $r \in I_p$ enters p along its drain, $r = t_{qi}$ for some $q \in S$ and i is even then $cw_p(r) = u_{pk}$. (Recall that $U_p = \{u_{p_1}, u_{p_2}, \ldots, u_{p_k}\}$).

The watershed graph $G = (V, E)$ for a nasty terrain has vertices

$$V = R \cup \bigcup_{p \in S} (U_p \cup \{u_{p0}\})$$

and edges

$$E = \bigcup_{p \in S} \left(\bigcup_{r \in R_p} (cw_p(r), r) \cup \bigcup_{t_i \in T_p \text{where } b_i \in R} (cw_p(t_i), b_i)) \cup \bigcup_{t_i \in T_p \text{where } b_i \in S} (cw_p(t_i), cw_{b_i}(t_i)) \right)$$

and abstracts the drainage characteristics of nasty terrains (Lemma 5.5.5). The convention that a path of steepest ascent t_i links to a single element of $S \cup R$ simplifies the notation for edge definitions.

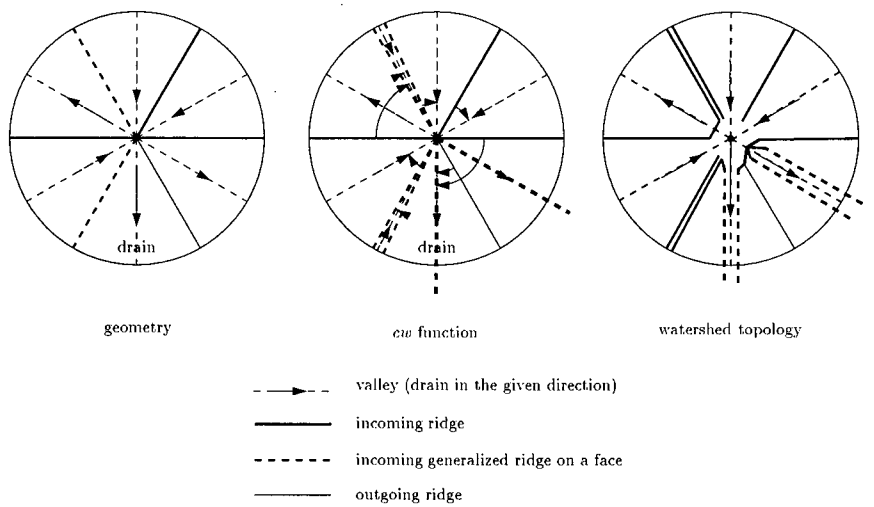


Figure 5.13: Watershed graph topology on a nasty terrain.

In the definition of the edges for a TIN point, the first union creates graph edges for adjacent TIN ridges, the second union creates graph edges for incoming generalized ridges whose paths of

steepest ascent end at ridges, and the third union creates graph edges for incoming generalized ridges whose paths of steepest ascent end at TIN vertices. Each union differs in the selection of the second endpoint for the graph edge.

Figure 5.14 shows a terrain (left) with ridges in bold lines and paths of steepest ascent across faces in dashed lines. In this figure, the interior for one of the watersheds is not simply connected since there are paths of steepest ascent that start from the centre saddle point and reach a saddle point on the left of the terrain. The corresponding watershed graph for this terrain appears in the right panel of the same figure.

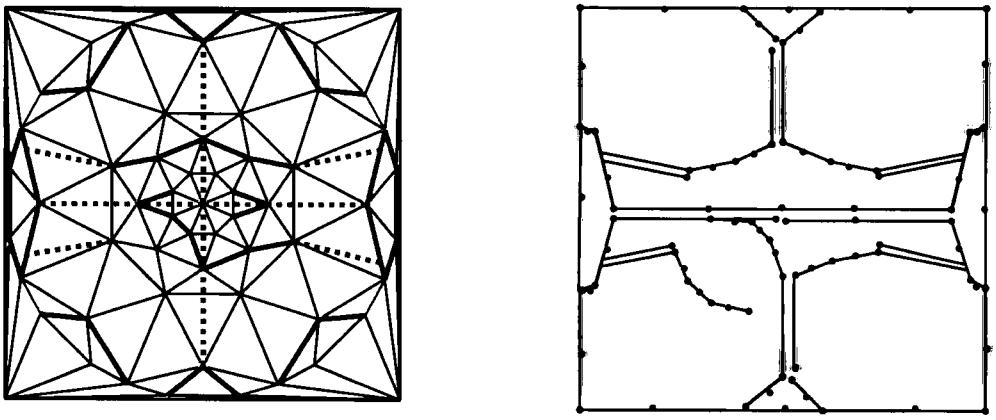


Figure 5.14: A nasty terrain (left) and its embedded watershed graph (right). Ridges appear as red edges and steepest ascent paths appear as dashed lines.

Lemma 5.5.5 *Let q be a point on a nasty TIN in a local neighbourhood N of TIN vertex p. There is a path in the embedded watershed graph between p and q that stays in N iff q drains to p or p drains to q.*

Proof: We concentrate on the drainage effects of paths of steepest ascent whose highest point is a saddle point of the TIN. The remaining topology of the watershed graph is identical to that of normal terrains and its drainage characteristics are covered by the proof of Lemma 5.5.3 for normal terrains.

Given a path of steepest ascent up that ends at a point p in the set S the path defines a pair of graph edges t_{2i+1} and t_{2i+2}. Only one such pair of graph edges can approach a point of S from each of its outgoing generalized valleys (Lemma 5.5.6). If the pair approach from

an outgoing generalized valley that is not the drain, then the cw_p function uses the same point of V as the endpoint of the edges, so the face is closed at p. If the pair approach from the drain of p then the cw_p function assigns u_{p0} as the endpoint of the most-counterclockwise edge and u_{pk} as the endpoint of the most-clockwise edge. The two edges close a face only if $k = 0$, which implies that there are no incoming generalized valleys at p and that p is a peak. ∎

Lemma 5.5.6 *At a point p in a TIN, at most one path of steepest ascent can reach p along each of its outgoing generalized valleys.*

Proof: Proof by contradiction. Suppose that two paths of steepest ascent A and B reach p along the same outgoing generalized valley. Then A and B follow one another as they leave p. Let q be the first point along the reversal of A and B where A and B differ. Since both paths are paths of steepest ascent, q must have more than one outgoing valleys. Any point with more than one outgoing valley, whether a TIN point or a point on a ridge edge, stops paths of steepest ascent, so both A and B should have stopped at q. This contradicts A and B ever reaching point p. ∎

Lemma 5.5.7 *The watershed graph for a nasty terrain is planar.*

Proof: The planar embedding of the watershed graph for nasty terrains is identical to the embedding for normal terrains of Lemma 5.5.4. Lemma 5.5.6 shows that only one pair of graph edges can ever reach a point along an outgoing generalized valley. If $t_{p,2i+1}$ and $t_{p,2i+2}$ reach a common graph point q then embedding $t_{p,2i+1}$ counterclockwise from $t_{p,2i+2}$ prevents the edges from crossing one another. The order of the edges at their common upper point is opposite from the order at their common lower point; edge t_{2i+1} is clockwise from edge t_{2i+2} at their lower endpoint and is counterclockwise from edge t_{2i+2} at their upper endpoint. With this embedding, the watershed graph remains planar. ∎

5.5.4 Watersheds of Arbitrary Terrain Points

The watershed graphs of Sections 5.5.1 to 5.5.3 concentrate on the watersheds for pits. More often we want to find the watersheds of other points, such as points on rivers. A simple extension of the watershed graphs for pits allows us to find the watersheds of arbitrary points.

The edges inside each face of the watershed graph create a monotonicity property for waterflows within the face.

- Let f be a face of a watershed graph that contains a pit p.
- Let $C_f : [0, 1) \rightarrow \mathbb{R}^2$ be the parameterised boundary of f in the watershed graph.
- Let in: $f \rightarrow [0, 2\pi)$ be the angle at which water placed at a point of f enters p.
- Assume that $q = C(0)$ is the most-clockwise point around C for which $in(q) = 0$.

From these definitions, Lemma 5.5.8 establishes the monotonicity property for the embedded watershed graph.

Lemma 5.5.8 *The internal edges of the watershed graph faces guarantee that if s and t are points of f where s is in the path of steepest descent from a point $s' = C_f(s_0)$ and t is in the path of steepest descent from a point $t' = C_f(t_0)$ and $s_0 < t_0$ then* $\text{in}(s) \leq \text{in}(t)$.

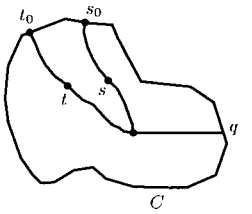


Figure 5.15: Drain order of internal and boundary points in a watershed

Proof: The order follows from Lemma 5.6.2 where there exists a path from each of s' and t' to the pit p that does not cross the boundary of f. The proof of the lemma uses the path of steepest descent from each of s' and t' to p. Since paths of steepest descent do not cross, the cyclic order of $in(s')$ and $in(t')$ around p does not change. By selecting $q = C_f(0)$ as the most-clockwise point around the boundary for which $in(q) = 0$, we ensure that $in(t')$ does not wrap through 0 degrees to be less than $in(s')$. Since the order holds for s' and t', it holds for any points along their paths of steepest descent. ∎

In other words, the order in which points drain into p is monotone in the order along the boundary C. As a consequence, if point q is in the watershed of point p then the watershed of q is contained in the watershed of p.

We use the monotonicity property and the fact that paths of steepest descent do not cross to find the watersheds of arbitrary points. Let p be an arbitrary point on the TIN. From the introduction to Section 5.4 we know that the paths of steepest ascent out of p locally delimit the land that drains through p. If we add these paths of steepest ascent from p into the watershed

graph and, among these added paths, connect in the watershed graph the two paths that bound the drain of p then we partition the face that contains p into the watershed of p and the rest of the watershed to which p drains. We can handle these changes to the watershed graph by treating p as if it were a saddle point of the graph and by closing the drain for p as a post-processing step.

We can classify the operations for an arbitrary point p as follows:

- p is on a face of the TIN: there is a single path of steepest ascent from p that stops at a ridge, a saddle, a peak, or a saddle. This path creates two edges in the watershed graph.
- p is on an edge of the TIN:
  - if p is on a ridge then p has no watershed
  - if p is on a valley then p has two paths of steepest ascent, one along each side of the valley, each of which creates two edges in the watershed graph
  - if p is on a transfluent edge then p has a single path of steepest ascent that creates two edges in the watershed graph
- p is on a vertex of the TIN:
  - if p is a saddle point then p is already in the watershed graph and we only close off its drain direction
  - if p has a valley edge then p is handled as a regular saddle point and will have two or more paths of steepest ascent
  - if p only has transfluent edges then p has a single path of steepest ascent that creates two edges in the watershed graph.

With these changes in mind, the watershed of a river simply becomes the watershed of the most-downstream point along the river.

5.5.5 A Simplified Watershed Graph for Pits

As shown in Lemma 5.5.8, the edges inside each face of a watershed graph enforce a monotonicity property: if points p, q, and r appear in counterclockwise order along a watershed boundary then this cyclic order occurs for the paths of steepest descent from p, q, and r at the pit to which they drain. These internal edges are crucial for identifying the watershed boundaries of arbitrary points

on the terrain. Unfortunately, these edges are also abundant in a detailed TIN, which makes it difficult to locate individual watersheds when we display all the graph edges for a TIN. For display, we only want to see the watershed edges that separate two faces; we want the watersheds of pits without the internal face structure.

The face simplification can occur at either of two locations: in the watershed graph or in the face point lists. In the watershed graph, we can remove any edge that does not belong to a cycle (Lemma 5.5.9). Standard graph theory algorithms can identify these edges [12]. In the face point lists, we can follow the sequence of points in a face and eliminate sequences that double-back on themselves to form a protrusion into the face. We can perform this second simplification with a simple algorithm that uses a stack to track the most-recently crossed points.. However, we want to keep spikes that emanate outward from a watershed since these spikes are part of the outer boundary characteristics of watersheds.

Lemma 5.5.9 *The watershed graph edges that belong to cycles are precisely the edges of the watershed graph that separate the watersheds of two pits.*

> **Proof:** Since every face of the watershed graph encloses exactly one pit (Lemma 5.6.4) the simplification does not remove any watershed boundaries between pits. Since every TIN point belongs to the same face of the watershed graph as the pit to which it drains (Lemma 5.6.2), no edge with the same face on both of its sides belongs to a cycle. The graph simplification leaves only the watershed boundaries between watersheds of pits. ∎

5.6 Correctness Proofs for Watershed Graphs

When describing the watershed graph for nice, normal, and nasty terrains, Lemmas 5.5.1, 5.5.3, and 5.5.5 prove that the watershed graphs preserve the local drainage characteristics of the terrain. The local proofs do not guarantee the global drainage properties that we need for identifying watershed boundaries.

At the beginning of Section 5.5 we presented some assumptions about the watershed of a point p:

- the watershed is one polygon,

- point p is in the watershed,
- the interior of the watershed is connected, and
- the watershed boundary does not self-intersect.

We then showed that the last two assumptions are inconsistent with even simple modelling assumptions. However, the watershed graph lets us prove that the first two assumptions are valid.

We begin by proving the second assumption, that each pit belongs in the same face of the watershed graph as the rest of its watershed, in Lemma 5.6.2. The proof is preceded by a short proof that paths of steepest ascent and steepest descent match one another under restricted conditions.

Lemma 5.6.1 *If p is in the path of steepest descent from q and there is no saddle between p and q in the path of steepest descent then q is in a path of steepest ascent from p.*

> **Proof:** Let γ be the path of steepest descent from q to p. Let π be the path of steepest ascent from the point where γ crosses an ϵ-ball around p. The hypothesis is that π and γ start in the same direction. If they diverge from one another then that point c has at least two incoming valleys: one along γ and one along the reversal of π. Therefore point c is a saddle point of the terrain (either on a valley edge or a TIN vertex). Point c must be a TIN vertex since π can only reach the interior of a valley edge by following the edge starting at the lower endpoint of the edge; and the π and γ paths will diverge at the lower endpoint just as they would in the middle of the edge. ∎

Lemma 5.6.2 *The path of steepest descent from a point q to the pit p to which q drains does not cross a face boundary in the embedded watershed graph.*

> **Proof:** Let G be the watershed graph for the TIN and let π be the path of steepest descent of q between q and p. We prove by induction that π never crosses an edge of G nor does it cross a face boundary at a vertex of G.
>
> Proceed by induction on the number of saddles that separate p and q along π. If π has no saddles then, by Lemma 5.6.1, path π and a path of steepest ascent from p are identical. Since paths of steepest ascent cannot cross, no other point around p generates an ascent path for G than can cross π.

Assume that π contains $n+1$ saddles ($n \geq 0$) and that s is the first saddle along π from q. Then π enters s either from a generalized incoming valley v or along a generalized incoming ridge r.

If π enters s along v and v is bounded by ridges t' and t'' then only one of $cw_s(t')$ and $cw_s(t'')$ is v so the watershed graph does not close the face that drains v and π can reach s without crossing an edge of the watershed graph.

Otherwise π enters v along r. Let r' and r'' be the two graph edges of r in the watershed graph. Conceptually, r' and r'' are slightly perturbed copies of r and q belongs to the valley between them. Again, only one of $cw_s(r')$ and $cw_s(r'')$ is r so π can reach s without crossing any watershed graph edge.

Finally, path π leaves s along its drain. By induction, if there is a path of steepest ascent that reaches s along its drain then this path corresponds to two watershed graph edges u' and u''. Only one of $cw_s(u')$ and $cw_s(u'')$ is the drain of s so s and π go between u' and u'' in G as required for our analysis with incoming ridge r above.

Inductively, path π does not cross any edges of G. Therefore p and q belong to the same face of G. ∎

There are three consequences of Lemma 5.6.2:

- every non-empty face of the watershed graph contains at least one pit,
- a pit belongs to the same face as the rest of its watershed, and
- the watersheds of the terrain are subsets of the faces of the watershed graph.

To establish that every watershed in the terrain is a face of the watershed graph, it is sufficient to show that every face contains at most one pit. Transitivity of path connectedness through Lemma 5.6.2 partitions the faces between the rest of the watershed points.

Lemma 5.6.3 *If points r and s are pits in the TIN then r and s belong to different faces in the watershed graph G.*

Proof: Proof by contradiction. Let π be a path from r to s that satisfies the following properties. The properties are listed in order of importance.

1. π lies in a single face of G

2. π has the lowest maximum elevation

3. π has the fewest contiguous sections of points at the maximum elevation

4. π has the shortest length of points at the maximum elevation

5. π has the shortest length

Let q be a point of maximum elevation on π. If the points of maximum elevation on π form a path, choose q as the first point encountered; π descends on at least one side of q.

Point q falls into one of three categories: q is on the interior of a TIN face, q is on the interior of a TIN edge, or q is a TIN vertex. We show that a contradiction arises in each case.

If q is on the interior of a TIN face f, let a and b be the points on the boundary for f where π enters and leaves the face. The path π with the section between a and b replaced by the line segment ab either has a lower maximum elevation (if q was the sole point of maximum elevation) or has a shorter length than π, which contradicts the conditions of π. Since path π traverses the face, there cannot be any paths of steepest ascent through the face for the new path to cross — the new path remains in one face of the watershed graph.

If q is on the interior of a TIN edge e, let a be the point where π enters one triangle adjacent to e and let b be the point where π leaves the other triangle adjacent to e. Let c be the endpoint with lowest elevation of e. Finally, let π' be the path π with the section between a and b replaced by the path acb. If q is the sole maximum elevation point then π' has a lower elevation than π. Otherwise, π' has fewer or a shorter length of points at the maximum elevation. Each of these cases contradict the conditions of π even though path π' may not be shorter in overall length than π. (The unfolding of the triangles adjacent to e could have aqb as a straight line and hence a shortest length path). Since path π crosses edge e, the edge cannot be a ridge. Therefore, there will not be any paths of steepest ascent that stop on e between q and c that path π' would cross. Path π' remains in one face of the watershed graph.

Finally, if q is a TIN vertex then it must be a support point of the faces of G: a saddle point or the endpoint of a ridge. If q is a saddle point then path π must belong to two separate sectors around q and π uses outgoing valleys since q has a maximum elevation. (If path π was only in one sector then we could just lower the elevation of π within the sector.)

Since π uses two different outgoing valleys, it must cross an edge of the watershed graph. ■

Theorem 5.6.4 *A face of the watershed graph has exactly one pit.*

Proof: Lemma 5.6.2 shows that every non-empty face contains at least one pit while Lemma 5.6.3 shows that no face contains two pits. ■

Although every face of the watershed graph is a single watershed of the terrain, we still need to trace the face boundaries. We simplify the tracing algorithm by proving that no face of the watershed graph is embedded inside another face.

Lemma 5.6.5 *A cycle of valley edges in a non-degenerate terrain contains at least one saddle point.*

Proof: Since we have a non-degenerate terrain, there are no horizontal edges. In particular, all the valley edges have a slope. Let p be the valley endpoint in the cycle of valley edges with highest elevation. Both valley edges adjacent to p have lower elevations than p while the triangles to either side of the valley edges have higher elevations than the valley edges, thus creating a local maximum in the height profile of p. Therefore, point p is a saddle point. ■

Lemma 5.6.6 *If water flows into a saddle point p from a sector S then p does not have an outgoing generalized valley in S.*

Proof: Since water enters p from sector S, sector S must be bounded by generalized incoming ridges. In the height profile function of the saddle point, only one local minimum can occur between the two adjacent local maxima that bound S and this is an incoming generalized valley. Therefore, the saddle point cannot drain through S. ■

Lemma 5.6.7 *If R_1 and R_2 are two components of the watershed graph then R_2 cannot be nested inside an internal face of R_1.*

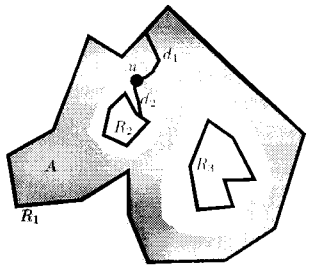


Figure 5.16: Nested components of a watershed graph for Lemma 5.6.7

Proof: Proceed by contradiction.

Let R_1, R_2, ..., R_n be components of the watershed graph with R_2, ..., R_n nested inside faces of R_1. Let A be the area inside R_1 to which R_1 exclusively drains: the portion of the path of steepest descent from a point ϵ away from R_1 that is shared only with the paths of steepest descent from other points near R_1. A is an open set. Let $\delta(A)$ be the boundary of A in R_1 and let u be the highest elevation point on $\delta(A) \setminus R_1$.

Since u is the highest point on $\delta(A) \setminus R_1$, the height profile of u has at least two outgoing generalized valleys along $\delta(A)$. The point u is a saddle. From the height profile function, generalized ridges (incoming or outgoing) separate the two outgoing generalized valleys from one another. An outgoing generalized ridge d_1 of u extends into A since part of R_1 drains near u. A generalized ridge d_2 extends out of A. Ridge d_1 leads to R_1 since it is in A.

If ridge d_2 is an incoming ridge then it either leads to a ridge component R_j or to a peak. But every peak has at least one adjacent ridge, and this ridge cannot belong to R_1 (otherwise the d_2 crossed $\delta(A)$). So, d_2 leads to an R_j inside R_1. By choosing ridges d_1 and d_2 to be neighbours across the side of $\delta(A)$ that isn't the drain for u, the two ridges d_1 and d_2 are connected in the watershed graph since they are separated by non-drain outgoing valleys. They join R_1 and R_j into one component, which contradicts R_j being a nested component.

Otherwise, ridge d_2 is an outgoing ridge. We can assume that all generalized ridges extending from u out of A are outgoing, otherwise the previous argument applies. Then the only water that enters u can come from A, so $u \in A$ and $u \in R_1$ by the connection d_1. Since $u \in R_1$, all the outgoing valleys around u belong to A, which contradicts $u \in \delta(A)$.

Therefore, R_1 cannot have unconnected components in an internal face. ∎

Since faces are not embedded within one another, we convert the watershed graph into a

connected graph by conceptually modifying the terrain so that the 2D bounding box for the terrain forms a quadrilateral of ridges around the terrain. The corners of this quadrilateral all drain away from the terrain. Since this bounding box surrounds all the other faces of the watershed graph, Lemma 5.6.7 proves that the entire graph must be connected. Therefore, any algorithm that follows one wall of a face will trace the whole boundary of the face without missing connections to holes in the face.

The watershed graph treats generalized ridges asymmetrically at saddles: incoming generalized ridges generate two graph edges while outgoing generalized ridges do not generate any graph edges. The graph edges for incoming ridges add meaningful structure to the faces that lets us find the watersheds of arbitrary points (Section 5.5.4). Lemmas 5.6.8 and 5.6.9 show that the outgoing generalized ridges do not perform the same role.

Lemma 5.6.8 *The paths of steepest ascent from pits do not affect the faces of the watershed graph.*

Proof: Since all local minima of the height profile function for a pit are incoming generalized valleys, the paths of steepest ascent from the pit are not closed in the watershed graph. Every sector around the pit belongs to the same face of the watershed graph. Therefore, adding these paths to the watershed graph simply makes a face-traversal algorithm reach a point p on the watershed boundary, descent to the pit, and then return to p to continue on the boundary. This path to the pit does not provide any additional information about the flow structure inside the watershed and can be omitted from the watershed graph. ∎

Lemma 5.6.9 *If r is an outgoing generalized ridge at a point p that crosses a face and r is not the drain of p then r either contains a point with more than one outgoing generalized valley or ends at a pit.*

Proof: Let s and t be points on either side of r near p. If s and t drain to the same point then r must drain to that point since paths of steepest descent do not cross. Therefore, r leads to a pit.

Otherwise, let π_s and π_t be the paths of steepest descent of s and t. These paths start with a fixed distance separating them but must diverge from one another since they end at different pits of the terrain. Let q_s and q_t be the points on π_s and π_t where π_s and

π_t begin to diverge from one another. Paths π_s and π_t cannot belong to the same face at q_s and q_t since they are starting to diverge and paths on a common face remain parallel. Therefore, there is a TIN vertex u between π_s and π_t where one of the paths starts onto a new face. Vertex u has more than one outgoing generalized valley, namely the direction that each of π_s and π_t take as they diverge. Since π_s and π_t can start arbitrarily close to r, vertex u must lie on r. ■

When the outgoing generalized ridge ends at a pit, the whole outgoing generalized ridge has no effect on the watershed boundary of the pit. When the outgoing generalized ridge contains a point with more than one outgoing generalized valley then that point produces paths of steepest ascent that end at p and complete the watershed graph.

5.7 Flat TIN Features

Throughout this work, we assume that the TIN has no horizontal edges and that all generalized ridges and valleys at TIN vertices are classified as incoming or outgoing. This assumption ensures that water flow is well-defined across the entire surface. However, horizontal edges do occur in TINs and we must at least mention some methods for handling them.

To resolve the water flow on flat features, we must expand our water flow assumption. When the terrain has no slope, there are several possible options for assigning the direction of flow for a point:

- flow towards the nearest river,
- treat the flat area (line or region) as a single vertex and flow towards the point along the boundary with the steepest descending slope, or
- flow towards the nearest point with a descending slope.

Each of these alternatives has appeared in the previous literature and thus is not handled in depth in this work.

Having water flow towards the nearest river corresponds to the approach of Skea et al. [81] in using Voronoi edges as watershed boundaries. Their approach assumes that the rivers are already provided; these rivers could come from an algorithm that detects drainage networks [21, 50, 58, 76, 87, 98] or could be manually traced. As long as the rivers are sufficiently dense, the Voronoi

edges partition flat features in a terrain-neutral manner. The value of the partition increases at river junctions. Flat triangles often appear at river junctions where TIN triangles have edges with one endpoint on each of the two rivers that are merging. In this case, the Voronoi edges generate a river divide between the two rivers. To combine with our algorithm, the Voronoi edges would not necessarily need to be embedded in the TIN. Critical TIN points, such as river junctions or points where a path of steepest ascent meets a flat triangle, would be matched with the Voronoi diagrams and Voronoi edges then play the role of paths of steepest ascent (across faces) in the watershed graph. Where a Voronoi edge reaches a point that once again has a distinct path of steepest ascent, that point is handled as a saddle point in the watershed graph.

Treating flat areas as a single vertex is the same as several depression-filling algorithms [52, 59, 57, 88]. The depression-filling algorithms flood shallow pits until they reach the lowest saddle point that is adjacent to the pit; the entire flat area then flows out of this lowest saddle point. The underlying assumption in these algorithms is that the drainage network forms a forest where each tree has its lowest point on the boundary of the current map. The medial axis of the flat feature [10] can act as a drainage network for the feature and lead the water to the single outlet. Using an artificial drainage network prevents us from assigning flow directions to triangles in a flat non-convex feature that lead the flow outside a reflex corner of the feature.

Allowing water to flow towards the nearest point with a descending slope allows flat features such as lakes to have more than one outlet [41]. Once again, the medial axis of the flat feature, or simply the Voronoi diagram of the TIN edges that bound the feature and have a descending path, provides a vector equivalent solution: the medial axis or Voronoi edges become ridges in the watershed graph and the Voronoi vertices become peaks in the TIN. Of course, these diagrams are restricted to just the flat feature. Where the medial axis or Voronoi edges meet an upwards slope, that point must be handled as a saddle point in the TIN.

5.8 Watershed Boundary Alternatives

Our algorithm for identifying watershed boundaries focuses on a single property of the TINs: paths of steepest descent do not cross each other. This property lets us follow the boundary of the watershed in parallel with a tour of the river network. There is another structure in the TIN that could also be exploited: the hierarchical nature of river networks and hence of watersheds.

The points in a river network where two river meet provide a natural hierarchical organisation of the rivers. River orders like the Strahler and Horton orders encode this hierarchy as river branching factors to distinguish important rivers from less important rivers. This hierarchy naturally extends itself to the river watersheds. In Figure 5.17, river r includes both rivers s and t. Similarly, the watershed for river r is the union of areas A, B, and C, where B and C are the watersheds for rivers s and t respectively, giving an inclusion hierarchy for the watersheds.

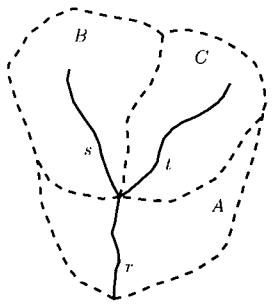


Figure 5.17: Hierarchy of rivers and their watersheds

On a TIN, the same merging of river courses that we see in rivers also occurs at saddles. At a saddle, all the water from each incoming generalized valley emerges from the single drain of the saddle just as rivers s and t emerge from their junction point along river r. Consequently, the watershed of a saddle point is the union of the watersheds for each incoming generalized valley. This union defines a two-level hierarchy of watersheds at the saddle point. When we apply this idea at every saddle point in the watershed of a pit, we obtain a complete hierarchy of watersheds for each pit in the terrain.

We can preprocess the hierarchy of watersheds to identify the watershed boundaries of points on the TIN. For each saddle point or river junction point, we would precompute the polygons whose area drains solely to that point along each incoming generalized valley. We would also precompute a graph of which polygon drains into which other polygon. In Figure 5.17, we would associate the most-downstream point of river r with polygon A and the junction point of rivers s and t with polygons B and C. Polygons B and C have graph edges to polygon A to indicate that they both flow into A. Then, given a point p on the TIN, we could locate the polygon D that contains p, subdivide D with at most two paths of steepest ascent from p, and output the half E of D that drains to p and the precomputed polygons that drain into E.

There are trade-offs for using a watershed hierarchy over using the watershed graph. The hierarchy of watersheds can precompute watershed polygons and create a point location structures that would allow us to quickly draw the watershed of a point. Drawing the interior of each precomputed polygon gives the impression of a connected watershed. However, the hierarchy does not lend itself as easily to providing a single polygon as the watershed of a point. To get a single polygon, the common boundaries of the precomputed watersheds must be eliminated or the precomputed

polygons must be spliced together at the saddle points. Since generating a single polygon for each watershed is a desirable approach, we opted for the watershed graph representation that uses a very simple boundary extraction algorithm.

5.9 Algorithm Complexity Analysis

We use a straight-forward algorithm, which we call the watershed graph algorithm, to construct the watershed graph of nasty terrains. The algorithm derives the vertices of the watershed graph from the vertices of the TIN, traces the paths on the TIN as graph edges, sorts the paths at each graph vertex, and defines the edge adjacencies for the watershed graph locally at each TIN vertex.

Lemma 5.9.1 *The watershed graph algorithm constructs the watershed graph of a TIN in* $O(k + n \log n)$ *time where* k *is the number of points in the embedding of the watershed graph and* n *is the number of vertices in the TIN.*

Proof: When identifying the vertices of the watershed graph, every TIN edge is considered once for the set R, and the profile function for every TIN vertex is considered once for the set S. Membership in R can be tested in constant time so R is found in $O(n)$ time. When considering the profile functions of the TIN vertices, every TIN triangle contributes to exactly three profile functions and every TIN edge contributes to exactly two profile functions, so over the entire TIN, all profile functions are constructed in $O(n)$ time. Each path of steepest ascent in T_p appears as an edge in the watershed graph so tracing these edges (hence defining the set T) takes $O(k)$ time across all TIN vertices. The paths of steepest ascent are sorted by vertex and by the angle that along which the path reaches the vertex. The sorting takes an additional $O(n \log n)$ time. The sorted paths of steepest ascent provide an order for constructing the watershed graph, which can be completed in $O(n)$ time since the topological adjacencies of the graph only depend on local properties of the height profile functions at each TIN point. The total complexity of constructing the watershed graph is then $O(nk + n \log n)$. ∎

Since a path of steepest ascent on a TIN can have $\Theta(n^2)$ points [26], k is $O(n^2)$ and the worst case time complexity for the algorithm is Θn^3.

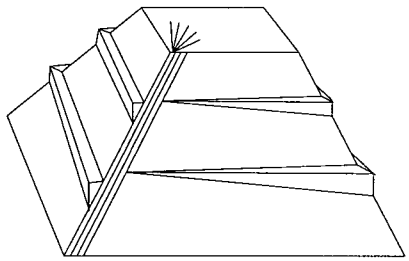


Figure 5.18: Terrain with $O(n^3)$ watershed complexity

Although the time complexity for the straight-forward algorithm is high, it is optimal: the watershed structure of a TIN can have $\Omega(n^3)$ complexity (Lemma 5.9.2). Most of the complexity comes from paths of steepest ascent with $O(n^2)$ points. Even if we assume that the paths of steepest ascent have $O(1)$ points, the straight-forward algorithm remains optimal. Under this condition, the complexity of the algorithm is $O(n \log n)$.

Lemma 5.9.2 *A watershed graph can have $\Theta(n)$ edges and $\Theta(n^3)$ points.*

Proof: The river complexity of a TIN with n edges can be $\Theta(n^3)$ with $\Theta(n)$ rivers [26] (Figure 5.18). The matching watershed boundary has $\Theta(n)$ edges and $\Theta(n^3)$ points. ∎

The watershed graph simplification algorithms of Section 5.5.5 each take linear time and linear space in their input size. For a TIN with n vertices, the watershed graph has $O(n)$ edges that may contain $\Theta(n^3)$ points. The watershed graph simplification algorithm takes the $O(n)$ edges as its input, without needing the $\Theta(n^3)$ embedding. For reasons of time complexity and spike preservation, we use the watershed graph simplification algorithm when simplifying graphs in Section 5.11.

5.10 Implementation Lessons

We implemented our algorithm for finding watershed graphs at the same time that we were developing the proofs of consistency between the graphs and the TINs. These concurrent activities benefited one another in many ways; we elaborate on five benefits in this section.

First, the implementation of the watershed graphs clarified exception cases that our initial proofs neglected. The first implementation computed watershed graphs that were not connected, which, in turn, caused the face-tracing algorithm to fail. This problem led us to prove that the watershed graph is connected only if we impose a boundary condition. The same connectivity

problem required us to clarify the behaviour at saddle points when a path of steepest ascent started along a face of the TIN.

Second, the implementation emphasized that we must concentrate more effort on combining data from different sources. In our case, we used the standard practice of embedding rivers as breaklines in the TIN to adjust the water flow characteristics on the TIN but did not account for elevation differences between the TIN and the rivers. In some cases, rivers would travel along the slope of a hill in the TIN, even after the rivers were embedded as breaklines—something unexpected in nature. The height differences between the elevation data and the river data motivated us to exploit the internal structure of the watershed graph for finding the watershed of an arbitrary point.

Third, maintaining the graph topology information separately from the edge embedding information simplified many implementation details. One of our ultimate goals was to display the watershed graphs. When many paths of steepest ascent reach a common ridge edge, one choice for representing the graph embedding could split the ridge edge into many small parts that join the ends of the steepest ascent paths. This choice favours the graph embedding rather than the graph topology. Instead, our implementation matched the graph representation with its topological description in Section 5.5.3 and stored the embedding information separately. We can then manipulate the $O(n)$ edges of the graph without incurring a potential $\Theta(n^3)$ cost for looking at all the points that embed the edges. This latter cost only occurs when we create the watershed graph or trace the faces of the graph.

Fourth, the TIN model for terrain captures a lot of terrain variation. Originally, we expected that there would be a relatively small percentage of saddle TIN vertices so the faces of the watershed graph would have little internal detail. During the implementation, we found that the faces of the watershed graph had a surprisingly dense internal structure. Since we can eliminate this internal structure when finding the watersheds of pits, the actual data suggest that we implement a simple algorithm to remove these internal edges to produce the cleaner watersheds of Figures 5.20 to 5.22.

Finally, the last lesson is one that is known to all implementors: even if the theory assumes that degeneracies do not occur, the implementation must still handle them. In our case, we had hoped that there would be few horizontal triangles in the mountainous regions of the TIN. We were surprised by the frequency of horizontal triangles, especially at river junctions.

5.11 Sample Watersheds

Figure 5.19 shows the watersheds for the mountains immediately north of Vancouver, British Columbia. The actual watersheds, according to the 1:50 000 scale Watershed Atlas of British Columbia, are the coloured polygons. In Figures 5.20 to 5.22, we show an embedding for the watershed graph with the bold lines. The watershed graph edges in these figures exclude much of the internal face structure. The terrain covers a 35 km by 35 km area and uses 30 500 points of the original 677 000 elevation data points. The accuracy and precision of the elevation points is discussed in Section 3.1. The elevations were converted into a TIN with a 40 metre error tolerance relative to the original data. The watershed graph for this terrain uses 46 525 edges and 88 077 points. The embedding for the watershed graph edges had at most 18 points per edge. The small size of the graph edges is encouraging for our algorithm given its $O(k + n \log n)$ complexity where k is the output size and n is the number of TIN vertices.

Although our goal is to eliminate inconsistencies in the watershed boundaries, the edges of Figure 5.20 differ from those in the Watershed Atlas of British Columbia. First, we have more watersheds in the interior of the mountains than reported by the Watershed Atlas. Remember that our algorithm finds the watersheds of the pits in the TIN. In some cases, the TIN has pits in the interior of the mountains caused by the error in the TIN relative to the terrain; our algorithm finds watersheds for these pits. We expect that smoothing the terrain, reducing the error tolerance of the TIN, and grouping the watersheds of pits that lie along a common river would improve the correspondence between the watersheds. In other cases, edges of the watershed graph converge to a point and seem to close off a face, but the edges are not topologically connected. Two faces in Figure 5.20 may be a single face in the watershed graph. Second, our watersheds contain lines that extend from the watershed boundaries into the watershed interiors. Since most of the internal structure is not represented in Figure 5.20, these additional edges link the outer boundary of a watershed to "holes" in the watershed that are caused by small pits in the terrain. Third, our watershed boundaries of Figure 5.20 do not always extend to the coast. Finally, some edges from the Watershed Atlas are missing. There are two reasons for this last discrepancy. In some cases, the TIN approximation to the terrain is poor and the drainage characteristic along the specified line is not clear. In other cases the Watershed Atlas defines the watershed of river tributaries separately from the main river. The watershed graph does not distinguish these tributary watersheds from

the main river watershed since it has no indication of the significance of the tributary.

The watershed graph of Figure 5.20 has 554 watersheds. These watersheds include areas of the ocean inlet on the bottom and left of the figure. Of these watersheds, 50 have an area below 1,700 m^2 and 3 have an area above 100 000 000 m^2 — these latter watersheds include parts of the ocean inlet. The remaining watersheds range in area from 2,257 m^2 to 91,646,904 m^2 with an average of 361,522 m^2.

A shortcoming of the watersheds in Figure 5.20 is that important tributaries of rivers are not assigned separate watersheds. We identified 22 such tributaries from the Watershed Atlas. By including the most-downstream point of these tributaries into the watershed graph, as described in Section 5.5.4, the graph separates the watersheds of these points from the watershed of the mainstem. These extra watersheds appear in Figure 5.21. By knowing about these tributaries, the results of the watershed graph in Figure 5.21 are closer to the watersheds of the Watershed Atlas than those of Figure 5.20.

Figure 5.22 shows the watersheds of the pits in the TIN with the same 40 metre error tolerance but with the rivers of Figure 5.19 embedded in the TIN as breaklines. While the presence of the rivers helps to clarify some of the watershed boundaries, the rivers also create horizontal edges in the TIN. In turn, these edges define extra pits in the terrain, each of which has its own small watershed in the watershed graph. These small watersheds are an undesirable artifact of the breaklines. In future comparisons, we should exploit the river breaklines more: if a known river exits a watershed then that watershed should merge with the downstream watershed. Exceptions would occur for special tributaries as in Figure 5.21. Under the current restrictions, we prefer the simpler watersheds of Figure 5.21 over the smaller ones of Figure 5.22.

Despite the differences between the boundaries in the Watershed Atlas and the boundaries in Figure 5.21, the watershed graph provides an advantage over the alternatives for transferring the existing 1:50 000 scale watersheds to the 1:20 000 scale maps. The watershed graphs adapt to terrain features at the 1:20 000 scale that do not appear in the 1:50 000 scale boundaries. This gives the new watersheds a better accuracy, with respect to the TIN, than the older 1:50 000 scale boundaries. The watershed graph is also more consistent and less tedious to construct than manually re-digitizing all the watershed boundaries from the 1:20 000 scale data.

Figures 5.23 and 5.24 compare the watershed boundaries of Figure 5.20 with the watersheds

from Jensen's raster algorithm [51]. The raster algorithm selects one pit and uses greedy hill climbing to define the watershed of the pit. In these two figures, we biased flat areas to flow towards the lower left corner since this is where the ocean inlet lies. The dark lines are the boundaries of the watershed graph and the coloured polygons are the raster watersheds.

For a common comparison, we generated the regular raster grids for Jensen's algorithm from the same TIN that was used for the watershed graph. The raster grids have one modification: we embed the coastline into the grid and raise the elevation of the coast by 20 metres so that the raster watersheds end more often on the coast and merge less with the ocean. Figure 5.23 has a 400×400 grid (one raster point every 90 metres) with 797 watersheds and Figure 5.24 has an 800×800 grid (one raster point every 45 metres) with 1566 watersheds. Unfortunately, we have to colour the raster watersheds in Figures 5.23 and 5.24 automatically so the watershed colours are not consistent between the figures and many adjacent watersheds have similar colours that do not transfer well to print.

The main candidate for a comparison with an existing vector algorithm is the system by Nelson et al. [66]. Since our underlying vector construction uses the same TIN partition lines as Nelson et al. and since the watersheds of Figure 5.21 do not contain any funnels as in Section 5.5.3, we would expect our results to match those of Nelson et al. on this TIN. However, we have not had the opportunity to implement Nelson's algorithm for a true comparison.

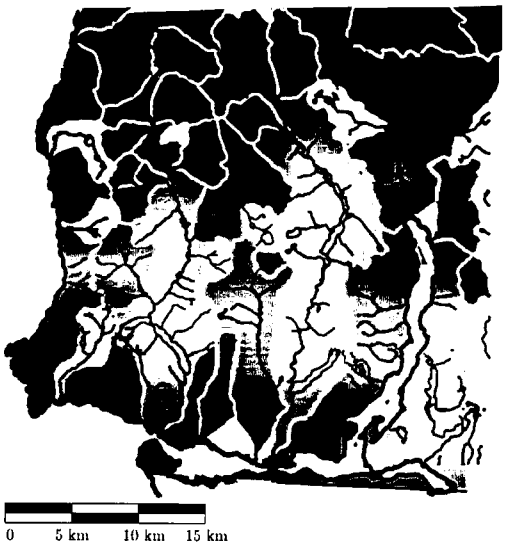


Figure 5.19: Rivers and watersheds from the BC Watershed Atlas of North and West Vancouver

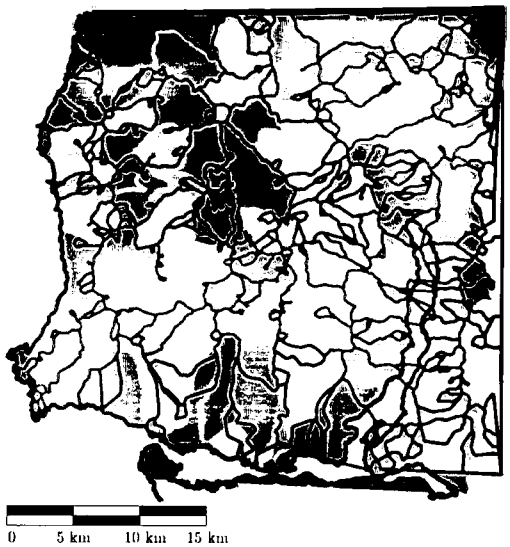


Figure 5.20: Derived (lines) and reference watersheds (shaded polygons) from the TIN

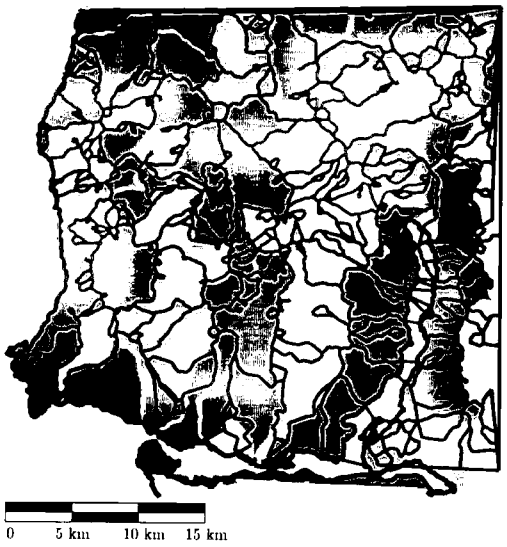


Figure 5.21: Derived (lines) and reference watersheds (shaded polygons) from TIN and selected river points

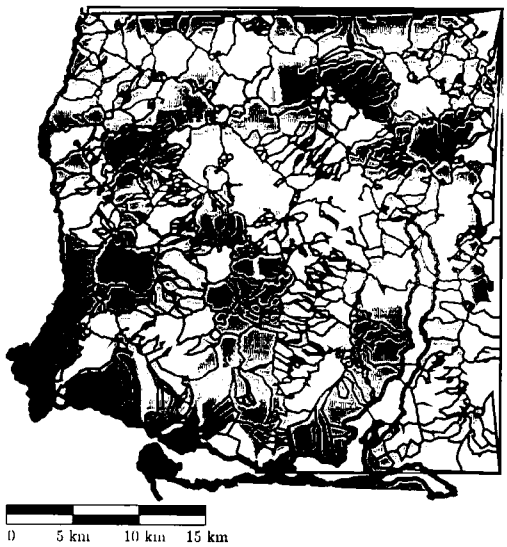


Figure 5.22: Derived (lines) and reference watersheds (shaded polygons) from the TIN with our rivers centrelines added as breaklines

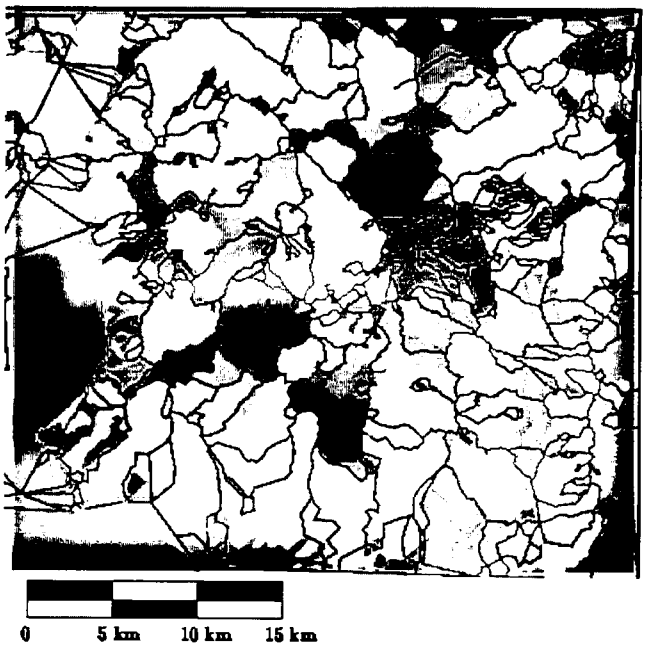


Figure 5.23: Derived (lines) and raster (shaded polygons) watersheds on a 400 × 400 grid

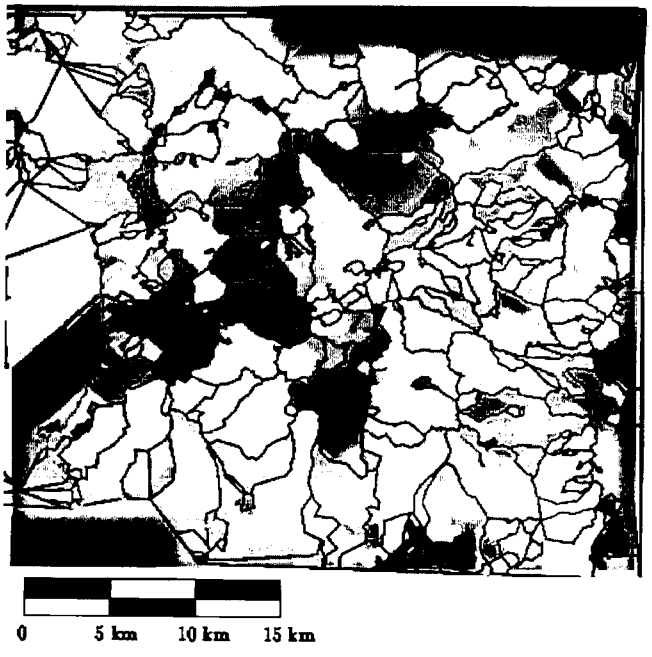


Figure 5.24: Derived (lines) and raster (shaded polygons) watersheds on a 800 × 800 grid

Chapter 6

Conclusion

The geometry in a problem often provides structure that we can exploit to find an efficient solution. The work in this dissertation finds implicit characteristics and consistent solutions in the geometry of two hydrology problems: simplifying river networks through centrelines and identifying watersheds from terrain elevations. We solve both problems in the domain of vector data since this is where the geometry of the problems is most readily accessible.

The medial axis of a river is commonly used as the centreline of a river or lake. However, the medial axis introduces two problems from a computational standpoint. First, it uses parabolic segments. Second, few robust implementations for finding the medial axis of a polygon exist. Consequently, we developed an algorithm that approximates the medial axis from a robust algorithm for the Voronoi diagram of points; the approximation is piecewise-linear. We also present an algorithm for orienting the edges of our approximation to induce a consistent flow of water in the network and characterize the cases where the orientation is not unique. Along with the approximation, we record the proximities between the centreline and the river banks or lake shores. The proximity information lets us derive additional river network characteristics:

- river area
- opposite river banks and lake shores
- pairings between bank and centreline attributes
- extensions of network orders to river banks and lake shores

Current vector algorithms for identifying watersheds from terrain elevations concentrate

on the area of the watershed. We adopt two fundamental assumptions: a triangulated irregular network (TIN) models the terrain and water follows the path of steepest descent on the terrain. With these assumptions, we demonstrate a terrain on which previous algorithms would either find only one part of the watershed or would report a watershed with two disconnected components. In analyzing the boundaries of watersheds, we classify TIN models into three types according to the geometric degeneracies that occur: nice, normal, and nasty terrains. The drainage characteristics of a nasty terrain is more complex than that of a nice terrain. When we think of watersheds on a TIN, we have certain expectations such as simple polygons with connected interiors and non-degenerate boundaries. Previous algorithm for finding watersheds search for the interior of the watershed and, by assuming that the interior is connected, generate one polygon for the watershed. We show that the interior of a watershed is only guaranteed to be connected in nice terrains while most TINs are normal terrains. Consequently, previous algorithms can miss some aspects of the watersheds. We describe a data structure called the watershed graph that represents the watersheds of the least constrained nasty terrains. Most importantly, our watershed graphs are proven consistent with the modeling assumptions. We show that a simple construction algorithm for the watershed graph is optimal in both the worst-case and expected-case.

Throughout our solutions, consistency is the underlying goal. River centrelines must be consistent with the river banks, river flows with existing drainage patterns, computed river area and opposite banks with the implicit river definitions. and watersheds with the modeling assumptions and the terrain. By maintaining the consistency, we can obtain algorithms that are simple, efficient, and robust.

6.1 Future Directions

The interplay between computational geometry and geographic information systems remains a rich area for research. For the river networks in this dissertation, we still want to compute river centrelines around islands and sandbars, and orient centrelines through braided rivers according to the rivers' areas in each branch or other factors such as the river network orders. A more challenging problem asks to maintain a river centreline as the river banks are simplified and prevent the river banks from crossing as a result of catastrophic simplification. With reliable river centreline networks, even simple graph theory algorithms, for shortest paths and reachability for example,

can open new analysis avenues for GIS users [94].

For watersheds, this dissertation uses very simple assumptions for surface water flow. Since we now have a consistent model for identifying watersheds, we can expand the assumptions to account for different types of soil, for vegetation, and for water inertia and volume, all of which can change the water flows on a terrain. We can also use known rivers to better eliminate small and spurious watersheds or to create a hierarchy of watersheds, ordered by watershed inclusion, to match the river hierarchies of mainlines and tributaries. With watersheds automatically associated with individual rivers, we can think of eliminating small rivers from a display by thresholding the area of the river watersheds. Finally, if we can trace changes in the terrain to the watershed graph, we can analyze the stability of the watershed boundaries as data points are added or removed from the terrain. Stable boundaries could indicate that mountain or ridgeline profiles are not changing while we simplify the terrain.

To benefit the computational geometry side, geographic information systems offer a variety of geometric problems such as polygon overlay, buffer computation around lines and polygons, and map generalization. While there exist efficient algorithms for some of these problems, we must evaluate the algorithms for robustness, degeneracy handling, and overall consistency of the solutions. Other GIS problems, such as map labeling, are NP-hard but must still be solved daily. The worst case behaviours may be infrequent in practice. Approximations and heuristics for problems such as cluster finding, proximity detection, object selection and amalgamation, and object displacement in map generalization all offer challenges for computational geometers to solve.

Bibliography

[1] A. Aggarwal, L. J. Guibas, J. Saxe, and P. W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Disc. & Comp. Geom.*, 4:591–604, 1989.

[2] N. Amenta. Directory of computational geometry software. http://www.geom.umn.edu/software/cglist/.

[3] D. S. Andrews. Simplifying terrain models and measuring terrain model accuracy. Technical Report TR-96-05, University of British Columbia, 1996.

[4] Association for Geographic Information. GIS dictionary. http://www.geo.ed.ac.uk/agidict/.

[5] F. Aurenhammer. Voronoi diagrams—A survey of a fundamental geometric data structure. *ACM Comp. Surveys*, 23(3):345–405, 1991.

[6] L. Band. Topographic partition of watersheds with digital elevation models. *Water Resources Research*, 22(1):15–24, 1986.

[7] B. G. Baumgart. Geometric modeling for computer vision. Technical Report STAN-CS-74463, Dept. Comput. Sci., Stanford Univ., Stanford, CA, Oct. 1974.

[8] H. Baumgarten, H. Jung, and K. Mehlhorn. Dynamic point location in general subdivisions. *J. Algorithms*, 17:342–380, 1994.

[9] L. Beineke and R. Wilson, editors. *Selected Topics in Graph Theory.* Academic Press, 1978.

[10] H. Blum. A transformation for extracting new descriptors of shape. In W. Wathen-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, 1967.

[11] J.-D. Boissonnat, O. Devillers, and M. Teillaud. A semidynamic construction of higher-order Voronoi diagrams and its randomized analysis. *Algorithmica*, 9:329–356, 1993.

[12] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications.* Macmillan, London, 1976.

[13] P. Bose, D. Bremner, and M. van Kreveld. Determining the castability of simple polyhedra. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 123–131, 1994.

[14] K. E. Brassel and R. Weibel. A review and framework of automated map generalization. *Intl. J. Geographic. Info. Systems*, 2(3):229–244, 1988.

[15] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991.

[16] B. Chazelle and M. Sharir. An algorithm for generalized point location and its application. *J. Symbolic Comput.*, 10:281–309, 1990.

[17] Z. Chen. Breaklines on terrain surface. In *Proceedings of GIS/LIS '88*, pages 781–790, 1988.

[18] L. P. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4:97–108, 1989.

[19] F. Chin, J. Snoeyink, and C.-A. Wang. Finding the medial axis of a simple polygon in linear time. In *Proc. 6th Annu. Internat. Sympos. Algorithms Comput.*, volume 1004 of *Lecture Notes Comput. Sci.*, pages 382–391. Springer-Verlag, 1995.

[20] R. Chorley, D. Malm, and H. Pogorzelski. A new standard for estimating drainage basin shape. *American Journal of Science*, 255:138–141, Feb. 1957.

[21] J. Chorowicz, C. Ichoku, S. Riazanoff, Y.-J. Kim, and B. Cervelle. A combined algorithm for automated drainge network extraction. *Water Resources Research*, 28(5):1293–1302, 1992.

[22] N. Chrisman. A transformational approach to gis operations. *International Journal of Geographical Information Science*, 13(7):617–637, 1999.

[23] S. Collins. Terrain parameters directly from a digital terrain model. *Canadian Surveyor*, 29(5):507–518, 1975.

[24] R. G. Cromley and G. M. Campbell. Noninferior bandwidth line simplification: Algorithm and structural analysis. *Geographical Analysis*, 23(1):25–38, 1991.

[25] R. G. Cromley and G. M. Campbell. Integrating quantitative and qualitative aspects of digital line simplification. *The Cartographic Journal*, 29(1):25–30, 1993.

[26] 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 Canad. Conf. Comput. Geom.*, pages 325–330, 1996.

[27] L. De Floriani and E. Puppo. A survey of constrained Delaunay triangulation algorithms for surface representaion. In G. G. Pieroni, editor, *Issues on Machine Vision*, pages 95–104. Springer-Verlag, New York, NY, 1989.

[28] D. Douglas. Experiments to locate ridges and channels to create a new type of digital elevation model. *Cartographica*, 23(4):29–61, 1986.

[29] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Canadian Cartographer*, 10(2):112–122, Dec. 1973.

[30] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley-Interscience, New York, 1973.

[31] G. Dutton. *A Hierarchical Coordinate System for Geoprocessing and Cartography*, volume 79 of *Lecture Notes in Earth Sciences.* Springer, 1999.

[32] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15(2):317–340, 1986.

[33] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9(1):66–104, 1990.

[34] H. Edelsbrunner and T. S. Tan. An upper bound for conforming Delaunay triangulations. *Discrete Comput. Geom.*, 10(2):197–213, 1993.

[35] Environmental Modeling Research Laboratory, Bringham Young University. WMS. http://www.ecgl.byu.edu/wms.htm.

[36] M. G. G. Foreman, C. B. James, M. C. Quick, P. Hollemans, and E. Wiebe. Flow and temperature models for the Fraser and Thompson rivers. *Atmosphere-Ocean*, 35:109–134, 1997.

[37] S. J. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[38] R. J. Fowler and J. J. Little. Automatic extraction of irregular network digital terrain models. *Computer Graphics (SIGGRAPH '79 Proc.)*, 13(2):199–207, Aug. 1979.

[39] A. U. Frank, B. Palmer, and V. B. Robinson. Formal methods for the accurate definition of some fundamental terms in physical geography. In *Proc. 2nd Intl. Symp. Spatial Data Handling*, pages 585–599, 1986.

[40] P. Fua. Fast, accurate and consistent modeling of drainage and surrounding terrain. *International Journal of Computer Vision*, 26(1):1–20, 1998.

[41] J. Garbrecht and L. Martz. The assignment of drainage direction over flat surfaces in raster digital elevation models. *Journal of Hydrology*, 193:204–213, 1997.

[42] N. K. Garg and D. J. Sen. Determination of watershed features for surface runoff models. *Journal of Hydrology*, 120(4):427–447, Apr. 1994.

[43] C. M. Gold, P. M. Remmele, and T. Roos. Voronoi diagrams of line segments made easy. In *Proc. 7th Canad. Conf. Comput. Geom.*, pages 223–228, 1995.

[44] L. J. Guibas, J. E. Hershberger, J. S. B. Mitchell, and J. S. Snoeyink. Approximating polygons and subdivisions with minimum link paths. *Internat. J. Comput. Geom. Appl.*, 3(4):383–415, Dec. 1993.

[45] L. J. Guibas, D. Salesin, and J. Stolfi. Epsilon geometry: building robust algorithms from imprecise computations. In *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, pages 208–217, 1989.

[46] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graph.*, 4(2):74–123, Apr. 1985.

[47] P. S. Heckbert and M. Garland. Fast polygonal approximation of terrains and height fields. Report CMU-CS-95-181, Carnegie Mellon University, 1995.

[48] J. Hershberger and J. Snoeyink. An $O(n \log n)$ implementation of the Douglas-Peucker algorithm for line simplification. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 383–384, 1994.

[49] R. E. Horton. Erosional development of streams and their drainage basins—hydrophysical approach to quantitative morphology. *Geological Society of America Bulletin*, 56:275–370, 1945.

[50] R. S. Jarvis. Drainage network analysis. *Progress in Physical Geography*, 1:271–295, 1977.

[51] S. Jensen. Automated derivation of hydrologic basin characteristcs from digital elevation model data. In *AutoCarto 7*, pages 301–310, 1985.

[52] S. Jenson and J. Domingue. Extracting topolgraphic structure from digital elevation data for geographic information system analysis. *Photogrammetric Engineering and Remote Sensing*, 54(11):1593–1600, 1988.

[53] M. P. Kumler. An intensive comparison of triangulated irregular network (TINs) and digital elevation models (DEMs). *Cartographica*, 31(2), 1994. monograph 45.

[54] T. Lang. Rules for the robot draughtsmen. *The Geographical Magazine*, 42(1):50–51, 1969.

[55] D. T. Lee and F. P. Preparata. Location of a point in a planar subdivision and its applications. In *Proc. 8th Annu. ACM Sympos. Theory Comput.*, pages 231–235, 1976.

[56] J. Little and P. Shi. Structural lines, TINs, and DEMs. In *8th International Symposium on Spatial Data Handling*, pages 627–636, 1998.

[57] D. Mark. Automated detection of drainage networks from digital elevation models. In *AutoCarto 6*, pages 288–298, 1984.

[58] L. Martz and J. Garbrecht. Numerical definition of drainage network and subcatchment areas from digital elevation models. *Computers & Geosciences*, 18(6):747–761, 1992.

[59] L. Martz and E. D. Jong. CATCH: A Fortran program for measuring catchment area from digital elevation models. *Computers & Geosciences*, 14(5):627–640, 1988.

[60] R. McMaster and S. Shea. Generalization in digital cartography. *Association of Americal Geographers*, 1992.

[61] R. B. McMaster. Automated line generation. *Cartographica*, 24(2):74–111, 1987.

[62] V. Milenkovic, K. Daniels, and Z. Li. Placement and compaction of nonconvex polygons for clothing manufacture. In *Proc. 4th Canad. Conf. Comput. Geom.*, pages 236–243, 1992.

[63] V. C. Miller. A quantitative geomorphic study of drainage basin characteristics in the clinch mountain area. Technical Report 3, Columbia University, 1953.

[64] L. R. Nackman and V. Srinivasan. Point placement for Delaunay triangulation of polygonal domains. In *Proc. 3rd Canad. Conf. Comput. Geom.*, pages 37–40, 1991.

[65] L. Najman and M. Schmitt. Geodesic saliency of watershed contours and hierarchical segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(12):1163–1173, 1996.

[66] E. J. Nelson, N. L. Jones, and A. W. Miller. Algorithm for precise drainage-basin delineation. *Journal of Hydraulic Engineering*, 120(3):298–312, Mar. 1994.

[67] B. G. Nickerson. Automated cartographic generalization for linear features. *Cartographica*, 25(3):15–66, 1988.

[68] J. F. O'Callaghan and D. M. Mark. The extraction of drainage networks from digital elevation data. *Comp. Vis. Graph. Image Proc.*, 28:323–344, 1984.

[69] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* John Wiley & Sons, 1992.

[70] O. L. Palacios-Velez and B. Cuevas-Renaud. Automated river-course, ridge and basin delineation from digital elevation data. *Journal of Hydrology*, 86:299–314, 1986.

[71] O. L. Palacios-Velez and B. Cuevas-Renaud. SHIFT: a distributed runoff model using irregular trangular facets. *Journal of Hydrology*, 134:35–55, 1992.

[72] J. Perkal. An attempt at objective line generalization. (Translated by R. Jackowski) Michigan Inter-University Community of Mathematical Geographers, Discussion Paper #10, 1966.

[73] T. K. Peucker and D. H. Douglas. Detection of surface specific points by local parallel processing of discrete terrain elevation data. *Computer Graphics and Image Processing*, 4:375–387, 1975.

[74] J. Pfaltz. Surface networks. *Geographical Analysis 8*, pages 77–93, 1976.

[75] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, NY, 1985.

[76] J. Qian, R. Ehrich, and J. Campbell. DNESYS—an expert system for automatic extraction of drainage networks from digital elevation data. *IEEE Transactions on Geoscience and Remote Sensing*, 28(1):29–45, Jan. 1990.

[77] D. Rappaport and A. Rosenbloom. Moldable and castable polygons. *Comput. Geom. Theory Appl.*, 4:219–233, 1994.

[78] A. Saalfeld. Delaunay edge refinements. In *Proc. 3rd Canad. Conf. Comput. Geom.*, pages 33–36, 1991.

[79] R. Seidel. Constrained Delaunay triangulations and Voronoi diagrams with obstacles. Technical Report 260, IIG-TU Graz, Austria, 1988.

[80] R. L. Shreve. Statistical law of stream numbers. *Journal of Geology*, 74:17–37, 1966.

[81] D. Skea, P. Friesen, J. Carr, A. Gallagher, I. Barrodale, E. Davies, F. Milanzzo, B. Corrie, and B. Noureddin. GeoData BC—TRIM watershed atlas. http://ssbux2.env.gov.bc.ca/srmb/twa_home.htm.

[82] B. E. Smith. *M. C. Escher: Landscapes to Mindscapes.* National Gallery of Canada, 1995. page 26, figure 29.

[83] A. N. Strahler. Quantitative analysis of watershed geomorphology. *Transactions of the American Geophysical Union*, 8(6):913–920, 1957.

[84] K. Sugihara and M. Iri. Construction of the Voronoi diagram for 'one million' generators in single-precision arithmetic. *Proc. IEEE*, 80(9):1471–1484, Sept. 1992.

[85] K. Sugihara and M. Iri. A robust topology-oriented incremental algorithm for Voronoi diagrams. *Internat. J. Comput. Geom. Appl.*, 4(2):179–228, 1994.

[86] Surveys and Resource Mapping Branch. *British Columbia Specifications and Guidelines for Geomatics. Digital Baseline Mapping at 1:20 000*, volume 3. Ministry of Environment, Lands, and Parks, Province of British Columbia, Canada, Jan. 1992. Release 2.0.

[87] L. Tang. Automatic extraction of specific geomorphological elements from contours. In *5th International Symposium on Spatial Data Handling*, pages 555–566, 1992.

[88] A. Tribe. Automated recognition of valley lines and drainage networks from grid digital elevation models: a review and a new method. *Journal of Hydrology*, 139(3):263–293, 1992.

[89] M. van Kreveld. Digital elevation models and TIN algorithms. In M. van Kreveld, J. Nievergelt, T. Roos, and P. Widmayer, editors, *Algorithmic Foundations of Geographic Information Systems*, number 1340 in Lecture Notes in Computer Science (tutorials), pages 37–78. Springer-Verlag, Berlin, 1997.

[90] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 212–220, 1997.

[91] M. Visvalingam and J. D. Whyatt. Line generalisation by repeated elimination of points. *Cartographic Journal*, 30(1):46–51, 1993.

[92] R. Weibel. Generalization of spatial data: Principles and selected algorithms. In *Lecture notes from CISM Advanced School on Algorithmic Foundations of Geographic Information Systems*, pages 99–152. Springer-Verlag, 1996.

[93] C. Westort. Generalized operators for sculpting digital topographic surfaces. In *7th International Symposium on Spatial Data Handling*, volume 2, page 8B.1, 1996.

[94] I. Williams, T. Brown, M. Foreman, S. Macdonald, M. McAllister, and D. Hawkins. A spatially explicit model to estimate effects of temperature, discharge, and spawning migration timing on early Stuart sockeye. In preparation, 1999.

[95] I. V. Williams, T. J. Brown, M. McAllister, and D. Hawkins. A spatially explicit stream and ocean network used to model habitat-salmon interactions. Technical Report 2294, Can. Tech. Rep. Fish. Aquat. Sci., 1999.

[96] G. Wolf. Hydrologic applications of weighted surface networks. In *5th International Symposium on Spatial Data Handling*, pages 567–579, 1992.

[97] C. K. Yap. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. *Disc. & Comp. Geom.*, 2:365–393, 1987.

[98] M. C. Zhang, J. B. Campbell, and R. M. Haralick. Automatic delineation of drainage basins within digital elevation data using the topographic primal sketch. *Mathematical Geology*, 22(2):189–209, 1990.