# LINE SIMPLIFICATION UNDER SPATIAL CONSTRAINTS

TITUS TIENAAH

June 2018

TECHNICAL REPORT
NO. 314

# LINE SIMPLIFICATION UNDER SPATIAL CONSTRAINTS

Titus Tienaah

Department of Geodesy and Geomatics Engineering
University of New Brunswick
P.O. Box 4400
Fredericton, N.B.
Canada
E3B 5A3

June 2018

## PREFACE

This technical report is a reproduction of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Geodesy and Geomatics Engineering, June 2018. The research was supervised by Dr. Emmanuel Stefanakis and Dr. David Coleman, and funding was provided by Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery and Engage Grants Programs.

Tienaah, Titus (2018). *Line Simplification Under Spatial Constraints*. Ph.D. dissertation, Department of Geodesy and Geomatics Engineering, Technical Report No. 314, University of New Brunswick, Fredericton, New Brunswick, Canada, 134 pp.

# Abstract

At the point of data collection, it is beneficial to capture linear features at the highest possible resolution to support various applications at different levels of detail. Line simplification is the process of selecting relevant characteristic vertices of a polyline based on some data reduction criteria. In the context of other planar objects, a line simplification must maintain a certain spatial and topological relations consistent with the original polyline. In computational graphics, data storage, network transmission, and spatial analysis, it is very beneficial to replace complex geometric objects with simpler ones that capture the relevant features of the original.

Given a polyline $L$ with $n$ vertices, the line simplification problem seeks to approximate $L$ as $L'$ with $m$ vertices, where $m < n$ based on some criteria. The goal is to find vertices of $L'$ that represent $L$ "well". In this dissertation, given the maximum separation between $L$ and $L'$ as $\varepsilon$ and a set of optional constraints, we focus on finding $L'$ as a subset of vertices of $L$ in the context of other arbitrary planar objects. Out of context simplification of $L$ can lead to topological errors. To constrain $L'$, we explore the following optional constraints: (i) $L'$ should preserve planar self-intersection and avoid introducing new self-intersections with itself or other polylines, (ii) consecutive segments of $L'$ should not invert the sidedness relation in $L$, (iii) $L'$ should preserve disjoint, intersect and minimum distance relation to other planar objects, and (iv) the sidedness of planar objects relative to $L$ should not change in $L'$.

Using a context based convergence towards the original polyline, we improve upon earlier methods by developing a set of geometric heuristics that precisely deforms parts of the original polyline to resolve topological conflicts. This novel approach prevents unnecessary vertex inclusion during reversion towards the original polyline. Algorithms developed in this dissertation are evaluated using real world datasets: road network, contour lines, and spatiotemporal trajectories. Experimental results show topological errors are avoided with competitive compression ratios relative to unconstrained simplification. The contributions of this dissertation are algorithms and tools for a consistent simplification of arbitrary polylines in the context of arbitrary planar objects.

# Acknowledgements

I would like to express my deepest gratitude to all the people who made this work possible. First, my deepest gratitude to my advisors: Dr. Emmanuel Stefanakis and Dr. David Coleman. Their support and encouragement provided me with many opportunities for research, teaching, attending conferences, and working on projects. I am very grateful for your help, financial support, and research guidance during my study at UNB.

I will also like to thank Dr. Monica Wachowicz and Dr. Sue Nichols, for reviewing my proposal and research support as part of my advisory committee. Thanks to Dr. Yun Zhang for his help in designing my minor examination. Special thanks to Dr. Brent Hall and Mr. Cameron Plouffe for their feedback and support during a collaboration between ESRI-Canada and UNB on an NSERC-Engage project.

Special thanks to all those who supported me during this research: Mrs. Sylvia Whitaker for her support and administrative help, my part-time employer at Resson Inc. Mr. Peter Goggin and Mr. Rishin Behl for financial support and work flexibility, Mr. Amir Abouhamzeh for his friendship and support, and to all my colleagues at Resson, I have advanced in my skill set as a spatial developer working alongside all of you, thank you!

# Table of Contents

# List of Tables

# List of Figures

# List of Symbols and Abbreviations

*Preamble*:

Let $L$ be a polyline with ordered set of $n$ segments.

Let $L'$ be the simplification of $L$ at $\varepsilon$. The coordinates of $L'$ are a subset of $L$.

---

| | |
|---|---|
| $\varepsilon$ | Error of approximating $L$ as $L'$ or maximum offset between $L$ and $L'$ |
| SED | Synchronized Euclidean Distance |
| GIS | Geographic Information Systems |
| GPS | Global Positioning System |
| BLG | Binary Line Generalization |
| ESRI | Environmental Systems Research Institute |
| ArcGIS | A GIS system by ESRI |
| VLSI | Very Large Scale Integration |
| VHR | Very High Resolution |
| LIDAR | Light Detection and Ranging |
| UAV | Unmanned Aerial Vehicle |
| RDP | Ramer-Douglas-Peucker Algorithm [1972-1973] |
| $Min-\varepsilon$ | Find $L'$ with at most $k$ vertices that minimizes $\varepsilon$ over all approximations of $L$ that have $k$ vertices |
| $Min-\#$ | Find $L'$ that uses the smallest number of vertices among all $\varepsilon$-approximations of $L$ given $\varepsilon > 0$ |
| $\overline{V_iV_j}$ | Generalized segment from vertices at index $i$ and $j$ of $L$; $V_i$ and $V_j$ are the vertices at $i$ and $j$ respectively |
| $\overline{P_iP_j}$ | Generalized segment from vertices at index $i$ and $j$ of $L$; $P_i$ and $P_j$ are the points at $i$ and $j$ respectively |

# Chapter 1: Introduction

"Everything should be made as simple as possible, but not simpler."

—Albert Einstein

Recent advancements in sensor technologies have lead to an increasing high resolution of spatial and temporal data collection. Sensors such as Light Detection and Ranging (LIDAR), Very High Resolution (VHR) Satellite Imagery (sub-meter panchromatic resolution), high resolution imagery and high definition video from unmanned aerial vehicles (UAV) have lead to detailed spatial and temporal mapping. Sensors equipped with Global Positioning System (GPS) and Radio-Frequency Identification (RFID) have also allowed outdoor and indoor tracking of moving objects.

Based on sensor and resolution of data capture, linear features and outlines of polygonal boundaries may be discretized at a certain level of detail. For example, topographic mapping agencies often collect aerial imagery at the highest resolution possible and then extract vector data by digitizing. To facilitate reproduction at various scales, methods of a raw vectorization often represent linear features or outlines of regions with more detail than is required [Douglas and Peucker, 1973; Gribov and Bodansky, 2004]. The results of vectorization usually need some form of compression or simplification.

Data reduction is often required to store, transmit, visualize and analyse high resolution data. Simplification is required when there is a change in map scale or when mixing multi-source data at different scales [Li and Openshaw, 1993]. During simplification, it is important to keep characteristic points of linear features and outlines of polygonal regions [White, 1985]. The graphic representation of linear features is common in many fields such as computer graphics, digital image processing, cartography, geographic information systems (GIS), very large scale integration (VLSI), and computational geometry. Based on a geometric context, a linear feature can be treated as a set of points, chain of lines (polyline), or a polygon (linear ring).

Line simplification is the process of selecting salient relevant vertices that capture the linear characteristics of the original polyline at some error bound [Weibel, 1997; McMaster, 1987]. If the simplification captures critical vertices of the original, great efficiency can be obtained in vector operations, speed of transmission over a network, storage, visualization, and resource minimization (e.g., VLSI circuit layout). A compact shape approximation can reduce computer resource (disk and memory) requirements and can accelerate computations involving shape information in finite element analysis, collision detection, visibility testing, shape recognition, and visualization [Heckbert and Garland, 1997].

In the context of other planar objects, out of context simplification can lead to topological errors [Muller, 1990; Guibas et al., 1993; de Berg et al., 1998; Saalfeld, 1999; Estkowski and Mitchell, 2001]. The complexity and hardness of the line simplification problem have been explored by Guibas et al. [1993], Estkowski [1998], Saalfeld [2000], and Estkowski and Mitchell [2001]. Existing methods lack a unified treatment of arbitrary polylines in the context of arbitrary planar objects while observing spatial and topological constraints.

This PhD dissertation presents a set of novel geometric heuristics for topological simplification of polylines under spatial constraints. The research started with contextual developments using the Ramer-Douglas-Peucker (RDP) algorithm [Ramer, 1972; Douglas and Peucker, 1973]. Using context based topological error resolution (convergence property of RDP), further advanced geometric heuristics were developed. The set of heuristics were then extended to contextual online trajectory simplification with support for ad hoc queries. Heuristics and algorithms developed were then used to extend ESRI ArcGIS cartographic toolbox to make practical this research in industry. This is an article-based PhD dissertation, presented and supported though the following papers:

1. Paper 1 (Peer Reviewed):

   **Tienaah, T.**, Stefanakis, E., & Coleman, D.(2015). Contextual Douglas-Peucker Simplification, Geomatica 69(3)327-338.

2. Paper 2 (In Review):

   **Tienaah, T.**, Stefanakis, E., & Coleman, D.(2018). Line Simplification While Keeping it Simple or Complex.

3. Paper 3 (In Review):

   **Tienaah, T.**, Stefanakis, E., & Coleman, D.(2018). Topologically Consistent Online Trajectory Simplification.

## 1.1 Dissertation Structure

This research is presented as a six chapter article-based dissertation. Chapter 1 is an introduction, a brief background, research problem, research objectives and scope of this dissertation. The next three chapters (2, 3 and 4) are peer reviewed journal papers, which are either published or under review at the time of drafting this document [Tienaah et al., 2015, 2018a,b].

In chapters 2, 3 and 4 the principal author conducted the primary research while the co-authors provided reviews and minor inputs on content and structure. Chapter 5 is a practical implementation of this research in a commercial GIS package. Chapter 6 is a summary of this research, limitations, open problems, and conclusion. Figure 1.1 lays out the structure of this dissertation.

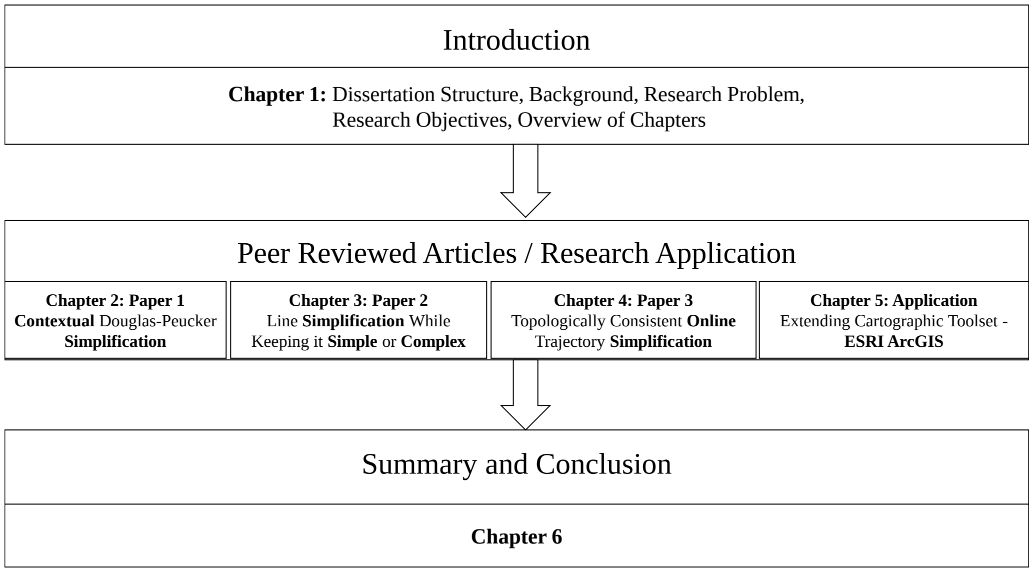

Introduction
**Chapter 1:** Dissertation Structure, Background, Research Problem, Research Objectives, Overview of Chapters
Peer Reviewed Articles / Research Application
**Chapter 2: Paper 1** **Contextual** Douglas-Peucker **Simplification**
**Chapter 3: Paper 2** Line **Simplification** While Keeping it **Simple** or **Complex**
**Chapter 4: Paper 3** Topologically Consistent **Online** Trajectory **Simplification**
**Chapter 5: Application** Extending Cartographic Toolset - **ESRI ArcGIS**
Summary and Conclusion
**Chapter 6**

Figure 1.1: Dissertation Structure

## 1.2 Background

A Linear features on a map is an abstraction of an areal object (river, road centerline, trails, boundary of a country, shoreline of a lake, contours of a terrain); in image processing, it is the edge or outline of regions in an image. The continuous motion of a moving object is often approximated as a trajectory. A trajectory at the point of data collection is often captured as a discrete set of consecutive positions with additional components such as time or speed. A moving object equipped with a Global Positioning System (GPS) can be person with a cellphone, GPS tagged animal (study of migratory or movement patterns), a GPS equipped navigating vehicle (car, vessel at sea, drone, air-plane). A high definition video of a moving object is also captured as discrete frames at a high frame rate. Each frame of a video is an image that captures the position of a moving object at a given time (slice of a time sequence).

The number of vertices used to represent the discrete representation of a liner feature or boundary of a region can vary at the point of data capture. Technological advancements in active and passive sensors indicate a trend towards high resolution data collection. For example, advancements in mapping technology such as Light Detection and Ranging (LIDAR) has made possible high-resolution terrain data (high resolution contours). Very High Resolution (VHR) satellite imagery such as WorldView-4 ($0.31m$) and Pleiades-1B ($0.5m$) are able to capture parts of the earth in sub-meter panchromatic resolution. In recent times, higher spatial resolutions are obtain through areal imagery by ultra high definition sensors on unmanned aerial vehicles (UAVs). To perform spatial analytics, raster to vector conversion is a common practice in many topographic mapping agencies. It is beneficial to vectorize linear features at the highest possible resolution and then resample it at different scales for various purposes.

High resolution linear representation leads to redundant shape or temporal information at the point of data capture. In geographic information systems (GIS), cartography, computer graphics, data compression or image processing, it is beneficial to replace complex geometric objects with simpler ones that capture the relevant features of the original [Buttenfield, 1985]. It is important to preserve linear characteristic vertices during data reduction as a set of representative features of the original. The process of selecting, displacing or introducing new vertices while removing other less critical vertices is line simplification [White, 1985; Weibel, 1997; McMaster, 1987; Saux, 2003; Guilbert and Saux, 2008].

### 1.2.1 Linear Features

In this dissertation, we consider an arbitrary linear feature as a polyline with an ordered chain of line segments. A line (segment) refers to a unidimensional extent joining two endpoints. A linear ring (polygonal) is formed if the first and last point in a polyline are coincident. The degree of a vertex in the chain of a polyline is the number of line segments incident on it [de Berg et al., 1998].

#### 1.2.1.1 Simple Polylines

A polyline is simple if consecutive segments only intersect at a connecting vertex (degree 2). In this dissertation, a simple polyline can be a segment with coincident endpoints (zero length) or with separate endpoints (non-zero length). A simple polyline can also be a chain of segments with separate or coincident endpoints (see Figure 1.2).

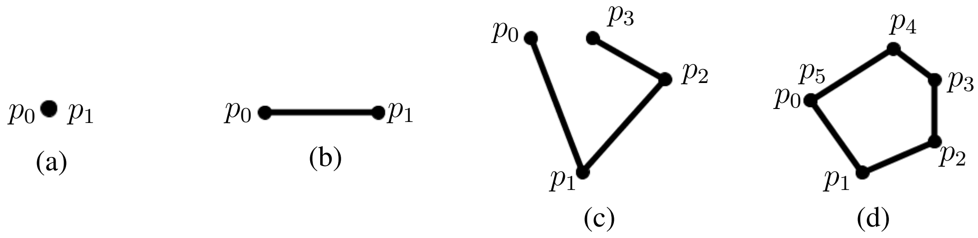


Figure 1.2: Simple Polylines: (a) segment of zero length, (b) line segment, (c) polyline and (d) ring (polygonal)

#### 1.2.1.2 Complex Polylines

A polyline may have planar or non-planar self-crossings. A planar self-intersection is a vertex with degree greater than two. Non-planar self-intersections have overlapping segments that intersect at a non-vertex or at collinear segments. See complex linear types in Fig.1.3.

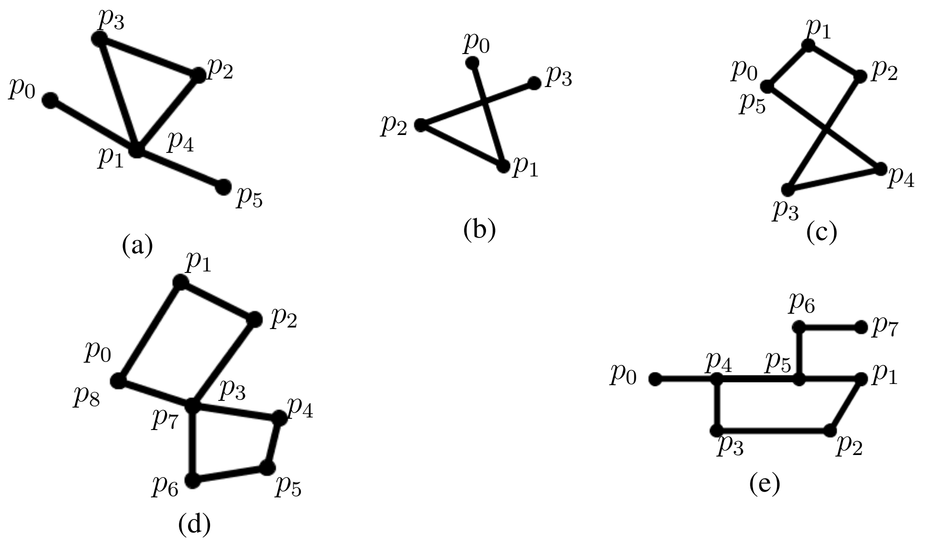


Figure 1.3: Complex Polylines: (a) planar self-intersection at $p_1$ (b) non-planar intersection between $\overline{P_0P_1}$ and $\overline{P_2P_3}$, (c) non-planar intersecting ring, (d) planar intersecting ring and (e) non-planar collinear intersection between $\overline{P_0P_1}$ and $\overline{P_4P_5}$

### 1.2.2 Line Simplification

Line simplification is an important operator in cartographic generalization. It involves a reduction of complexity in a map, emphasizing the essential while suppressing the unimportant, maintaining logical and unambiguous relations between map objects, and preserving aesthetic quality [Weibel, 1997]. In manual generalization, the important features of a line are selected, simplified, exaggerated, smoothed or displaced from other features [McMaster, 1987]. These features known as “critical points” include those which are related to the physical characteristics of a line and those related to a “perceived” sense of relevance, such as vertices closer to a city along a river [White, 1985]. In effect, it is a subjective cognitive process.

Most simplification algorithms seek to remove redundant coordinates from a polyline by first finding salient “features” of the polyline. Selection of critical points is based on the topological relationship between points and their neighbours, the extent of this neighbourhood search varies based on algorithm [McMaster, 1987]. Smoothing algorithms on the hand displaces points in attempt to reduce angularity thereby giving a flowing appearance to a polyline [Saux, 2003; Guilbert and Saux, 2008]. For a detailed survey of line simplification algorithms and their development over the years, refer to Weibel [1997], Heckbert and Garland [1997], and Shi and Cheung [2006].

Different types of polylines express various topological characteristics within itself or with other planar objects (points, lines and polygons). The process of line simplification introduces positional error in the resultant polyline and may cause topological errors [Muller, 1990; Saalfeld, 1999; Estkowski and Mitchell, 2001; da Silva and Wu, 2006; Abam et al., 2014; Arge et al., 2014; Funke et al., 2017]. In the context of other planar objects, the spatial relationship between the simplified polyline and objects in its planar space may be different compared to the original polyline. Topological conflict resolution for different types of polylines is at the core of the line simplification problem.

#### 1.2.2.1 Categorization of Algorithms

McMaster [1987] classified line simplification algorithms into various categories [Weibel, 1997]:

- **independent point**: e.g., every $n^{th}$ point of a polyline is selected, the others are removed,
- **local processing**: e.g., vertices are selected based on euclidean distance between consecutive vertices, the perpendicular distance to a base line connecting the neighbours of a vertex, or angular change,
- **constrained extended local processing**: these algorithms search beyond immediate vertex neighbours e.g., Lang [1969],
- **unconstrained extended local processing**: e.g., Reumann and Witkam [1974], and
- **global routines**: e.g., Douglas and Peucker [1973]; Visvalingam and Whyatt [1993].

Various simplification algorithms in these categories can further be classified into two main variants: *unrestricted* and *restricted* [Agarwal and Varadarajan, 2000; Abam et al., 2014]. In the *unrestricted* case, the vertices of the simplification are allowed to be any arbitrary points, not just the vertices of the original polyline (see Imai and Iri [1986]; Guibas et al. [1993]; Goodrich [1995]; Saux [2003]; Guilbert and Saux [2008]; Goethem et al. [2013]). In the restricted version, the vertices of simplified polyline are a subsequence of the original.

Whether *restricted* or *unrestricted* every line simplification algorithm introduces a deviation from the original, the error ($\varepsilon$) of the simplification [Cromley, 1991]. Simplification algorithms are also classified based on their error criteria. Some of the error criteria in literature are: bandwidth or parallel strip error criterion, Hausdorff distance, Fréchet distance, areal displacement, and vector displacement [de Berg et al., 1998].

The scope of this dissertation is limited to the *restricted* line simplification problem. To preserve the topological characteristics during simplification, this dissertation uses variants of the Douglas-Peucker (DP) or the Ramer-Douglas-Peucker Algorithm (RDP) algorithm based on a tolerance bandwidth error ($\varepsilon$) criterion [Ramer, 1972; Douglas and Peucker, 1973].

RDP is a restricted (subset property) global line simplification algorithm. The algorithm has been independently proposed by Ramer [1972] in image processing and Douglas and Peucker [1973] in cartography. RDP is easy to implement and its recursive nature lends to a hierarchical structure for multi-scale simplification [Cromley, 1991; Jones and Abraham, 1987; Van Oosterom, 1991; Tienaah et al., 2015]. It has a proximity property that ensures a simplification is within a distance $\varepsilon$ of the original polyline. $\varepsilon$ may be increased or decreased at any step in the algorithm to allow for different levels of pruning. This allows for any level of precision to the original polyline by choosing a small enough $\varepsilon$ as a means of converging to the original [Saalfeld, 1999]. In this dissertation, we exploit these properties to resolve spatial and topological errors.

White [1985] performed a study on critical points of a linear feature as a psychological measure of curve similarity; she reported the RDP method was best at choosing critical points compared to other methods ($n^{th}$ point elimination and selection through a perpendicular calculation between three consecutive points). A study of mathematical similarity and discrepancy measures between a polyline and its simplification by McMaster [1987] ranks RDP as "mathematically superior". The RDP algorithm is also one of the most popular simplification algorithms in open-source and commercial GIS, databases, and software tools [Tienaah et al., 2015]. RDP is used as a basis for polyline simplification in this dissertation to make practical the results of this research in academia and industry.

### 1.2.3 RDP Algorithm

Let the coordinates ($V_L$) of a polyline($L$) be $P_0, P_i, ..., P_n$ where $P_0$ is $(x_0, y_0)$ and $P_n$ is $(x_n, y_n)$. $L$ is defined by $n + 1$ coordinates and $n$ line segments: $\overline{P_0P_1}$, $\overline{P_1P_2}$, $\overline{P_2P_3}$, ..., $\overline{P_{n-1}P_n}$. $\varepsilon$ is the maximum separation (simplification error) between $L$ and $L'$, where $L'$ is the simplification of $L$ at $\varepsilon > 0$. The vertices of $L'$ ($V_{L'}$) are a subset of $L$ ($V_{L'} \in V_L$).

The algorithm starts by first creating an approximation of the entire (global) polyline as $\overline{P_0P_n}$. It then computes the distance offset of each vertex from $P_{0+1}$ to $P_{n-1}$. In this dissertation, the distance offset is the minimum Euclidean distance from a vertex $P_x$ to $\overline{P_0P_n}$; the original RDP by Douglas and Peucker [1973] uses a perpendicular distance, which is not always the minimum distance from a point to a segment. Let $P_k$ be the vertex with maximum of the minimum offset distances from $\overline{P_0P_n}$. If $\varepsilon_k > \varepsilon$, split $L$ into two sub-polylines $L_{0,k}$ ($P_0, P_1, P_i, ..., P_k$) and $L_{k,n}$ ($P_k, P_{k+1}, P_j, ..., P_n$), where $\varepsilon_k$ is the offset distance from $P_k$ to $\overline{P_0P_n}$. The algorithm recursively simplifies $L_{0,k}$ and $L_{k,n}$ as new inputs. The recursion terminates if $\varepsilon_k \leq \varepsilon$ or $L_{i,j}$ is a segment ($j - i = 1$, where $j > i$). $V_{L'}$ is a set composed of $P_0$, $P_n$ and the endpoints of each recursive sub-polyline. The algorithmic complexity of RDP is $\mathcal{O}(n^2)$.

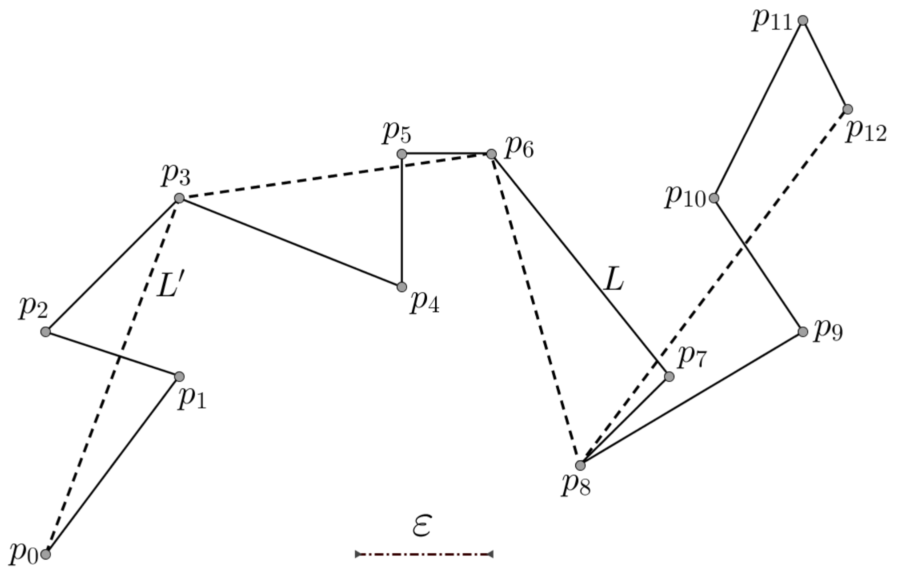


Figure 1.4: RDP Simplification

Figure 1.4 shows simplification of $L$ as $L'$ at $\varepsilon$. The minimum offset distance ($\varepsilon_{11}$) from $P_{11}$ to the generalized segment (dashed line) $\overline{P_8 P_{12}}$ is the length of $\overline{P_{11} P_{12}}$ which is not perpendicular to $\overline{P_8 P_{12}}$. The ability to change the error function of RDP makes it suitable for extension to spatiotemporal trajectory simplification [Keogh et al., 2001; Meratnia and Rolf, 2004].

#### 1.2.3.1 Spatial and Topological Relations

Using selected characteristic vertices by the RDP algorithm, a context planar object of a sub-polyline $L_{i,j}$ (from index $i$ to $j$) is any object ($C_i$) that intersects the convex hull (grey region) of $L_{i,j}$ or at certain distance ($\delta$) from its boundary. Consider $C_A$ and $C_B$ in Figure 1.5, $C_A$ is a context object of $L_{3,6}$ and can a context object of $L_{6,8}$ based on a user defined $\delta$; $C_B$ is a context neighbour of $L_{6,8}$.

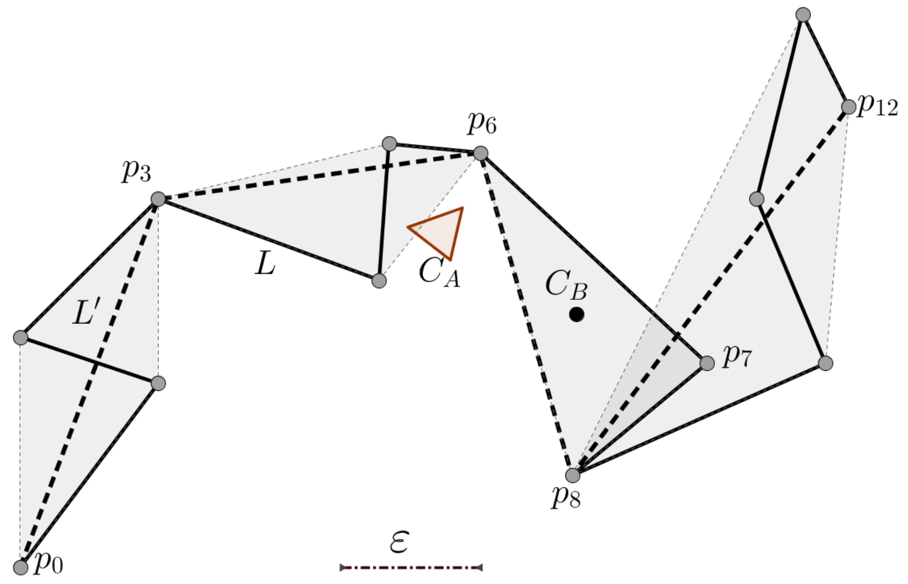


Figure 1.5: RDP Context Neighbours

In this dissertation, the following spatial and topological relations are investigated when deforming sub-polylines of $L$.

1. **Disjoint / Intersect**: $L'_{i,j}$ should express intersect or disjoint relation to $C_i$ as expressed by the sub-polyline $L_{i,j}$.

2. **Proximity**: $L'_{i,j}$ should express a certain user defined minimum distance to any $C_i$ as expressed by $L_{i,j}$. This relation is relaxed ($L'_{i,j}$ can violate the constraint) if $L_{i,j}$ violates the proximity constraint.

3. **Homotopy / Sidedness**: the sidedness of $C_i$ in relation to $L_{i,j}$ should not change when deformed as a segment $\overline{P_iP_j}$. Keeping the endpoints ($P_i$ and $P_j$) of $L_{i,j}$ fixed, $L_{i,j}$ is homotopic to $\overline{P_iP_j}$ if $L_{i,j}$ can be deformed into a segment $\overline{P_iP_j}$ without leaving (stepping over $C_i$) the planar space [Cabello et al., 2004]. In Fig. 1.5, keeping $P_6$ and $P_8$ fixed, $L_{6,8}$ cannot be collapsed as $\overline{P_6P_8}$ because of the presence of $C_B$ between $L_{6,8}$ and $\overline{P_6P_8}$. A further deformation of $L_{6,8}$ is required using the RDP algorithm, in this example, $L_{6,8}$ will split as $L_{6,7}$ and $L_{7,8}$. Note that by splitting $L_{6,8}$ as $L_{6,7}$ and $L_{7,8}$, $\overline{P_8P_{12}}$ is now topologically invalid: segment $\overline{P_6P_7}$ introduces a self-intersect

with $\overline{P_8 P_{12}}$ and the chain $\overline{P_7 P_8 P_9}$ makes a counter-clockwise turn at $P8$ whereas $\overline{P_7 P_8 P_{12}}$ makes a clockwise turn at $P8$. To resolve these topological errors, $L_{8,12}$ needs further deformation using RDP.

## 1.3 The Line Simplification Problem

Line simplification also known as polygonal, chain, minimum-link, subdivision or piece-wise linear approximation is a well studied problem. Given a polyline $L$ with $n$ segments, the line simplification problem seeks to approximate $L$ as $L'$ with $m$ vertices, where $m < n + 1$. The goal is to find vertices of $L'$ that represent $L$ with some "fidelity". The simplification criteria for computing $L'$ can be categorized into two forms [Kurozumi and Davis, 1982; Imai and Iri, 1988; Chan and Chin, 1992]:

- the $Min{-}\varepsilon$ problem: computes $L'$ with at most $K$ vertices that minimizes $\varepsilon$ over all approximations of $L$ that have $K$ vertices, and
- the $Min{-}\#$ problem: computes $L'$ that uses the smallest number of vertices among all $\varepsilon$-approximations of $L$ for $\varepsilon > 0$.

In this dissertation we focus on the *restricted* $Min{-}\#$ problem in the context of other planar objects (points, lines, and polygons).

## 1.4 Related Work

Imai and Iri [1988] present a unified approach to simplification of simple polygonal chains by formulating it in terms of graph theory. They considered the $Min{-}\varepsilon$ and $Min{-}\#$ problems under various error criteria. Chan and Chin [1992] improves the $Min{-}\#$ problem by Imai and Iri [1988] from $\mathcal{O}(n^2 \log n)$ to $\mathcal{O}(n^2)$. Chan and Chin [1992] further demonstrates that if the polyline to be approximated forms part of a convex polygon, the time complexity can be reduced to $\mathcal{O}(n)$. Other improvements on the work of Imai and Iri

[1988] have been explored by Melkman and O'Rourke [1988]. Agarwal and Varadarajan [2000] notes that $Min{-}\#$ and $Min{-}\varepsilon$ simplification problems can become intractable if the approximation is required to be simple or homotopic to the original chain.

One of the requirements in a map subdivisions is that polylines are required to be simple. No two edges of the subdivision may intersect, except at endpoints [de Berg et al., 1998; Estkowski, 1998]. de Berg et al. [1998] indicates the following algorithms do not constrain their outputs to be simple: Chan and Chin [1992], Douglas and Peucker [1973], Li and Openshaw [1992] and Melkman and O'Rourke [1988]. de Berg et al. [1998] builds on the work of Imai and Iri [1988] using points as planar context objects in subdivision simplification. Their implementation avoids self-intersection and intersection with other polylines of the sub-division by temporally adding all vertices of other polylines as extra points to the contextual set. Corcoran et al. [2011] indicates the proposal by de Berg et al. [1998] is only true if and only if the polyline being simplified is monotone.

In Figure 1.6, let $L$ and $L'$ be two simple polylines oriented from vertex $P_0$ to vertex $P_n$, and let $C$ be a set of planar context points. The approach to topological consistency in the context of $C$ by de Berg et al. [1998] is based on the idea that there exist a chain $L_s$ that completes both $L$ and $L'$ as simple polygons denoted by $L_sL$ and $L_sL'$ respectively [Corcoran et al., 2011; da Silva and Wu, 2006]. $L$ is topologically consistent to $L'$ if $L_sL$ and $L_sL'$ contain the same subset of points in $C$ otherwise it is inconsistent. In the inconsistent case, a point $c$ in $C$ lies between the bounded face of the polygon formed by $L$ and $L'$. Note that the polygon formed by $L$ and $L'$ can be complex though $L$ and $L'$ are simple. Corcoran et al. [2011] provides a proof for this topological sidedness relation using a point-in-polygon test. For $c$ in $C$, if $I(c, L) + I(c, L') \mod 2 = 0$ then $c$ is outside else it is inside the bounded face formed by $L$ and $L'$ (even-odd rule algorithm [Shimrat, 1962]); $I$ is the ray casting function.

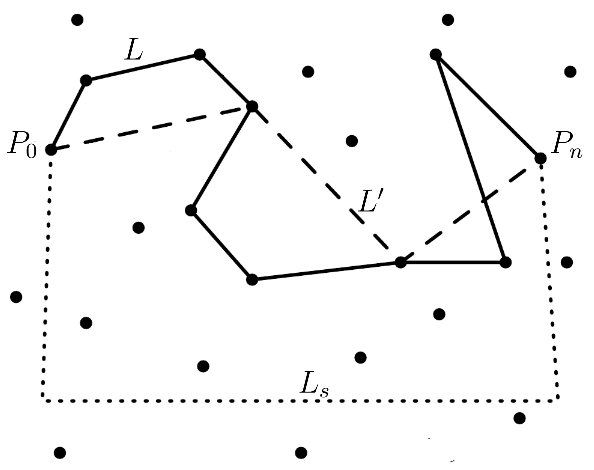


Figure 1.6: Topological consistency of $L$ to $L'$ in the context of $C$ (adapted from da Silva and Wu [2006])

de Berg et al. [1998] propose their monotone algorithms for determining planar topological consistency can be generalized to arbitrary chains by applying it to sub-chains which do not cycle or contain backward tangents. Intuitively, no backward tangent means that if $P_i$ is the first vertex of a chain, there should not be an edge $\overline{P_j P_{j+1}}$ that is closer to $P_i$ than the preceding edge $\overline{P_{j-1} P_j}$ [de Berg et al., 1998]. The drawback of this approach to topological consistency is that it may only be applied to a subset of possible simplifications [Corcoran et al., 2011]. Another disadvantage of this approach is that the context object set ($C$) grows based on number of neighbours to the polyline being simplified.

Saalfeld [1999] proposed to apply the strategy for monotone chains by de Berg et al. [1998] to arbitrary chains. Let $L$ be a simple arbitrary chain and $L'$ its simple simplification. $P_0$ and $P_n$ are the endpoints of $L$. Let $C$ be a set of planar context points. Corcoran et al. [2011] attempts to show that the proposal by Saalfeld [1999] is well defined for monotone and arbitrary simple chains: $L'$ is a consistent simplification of $L$ with respect to $C$ if and only if no point of $C$ lies in the bounded face formed by $L$ and $L'$ [Corcoran et al., 2011]. This is illustrated in Fig. 1.7, context points $O_1$ and $O_2$ are in the bounded face (grey region) formed by $L$ and $L'$ using the point-in-polygon test.

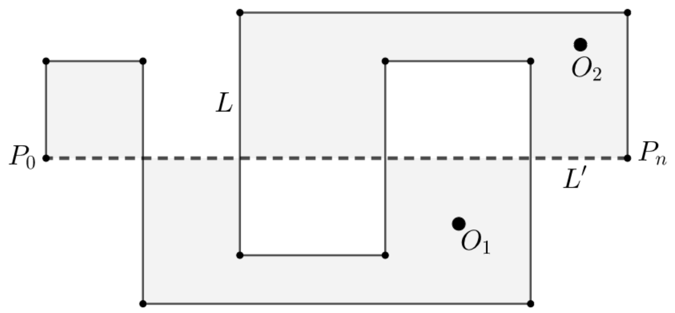


Figure 1.7: Contextual topological consistency in the bounded face (grey region) of $L$ and $L'$

A point-in-polygon test and proof provided by Corcoran et al. [2011] is not sufficient for determining if $L'$ is a consistent simplification of $L$ with respect to points in $C$. For example, in Fig. 1.8, consider bounded face formed by two simple polylines $L$ and $L'$, where $L'$ is the simplification of $L$ with context points $O_1$ and $O_2$ in $C$. The point-in-polygon algorithm will report $O_1$ and $O_2$ are outside the bounded face formed by $L$ and $L'$. Keeping the endpoints ($P_0$ and $P_n$) of $L$ fixed, $L$ cannot be deformed into $L'$ in the context of $O_1$ and $O_2$.

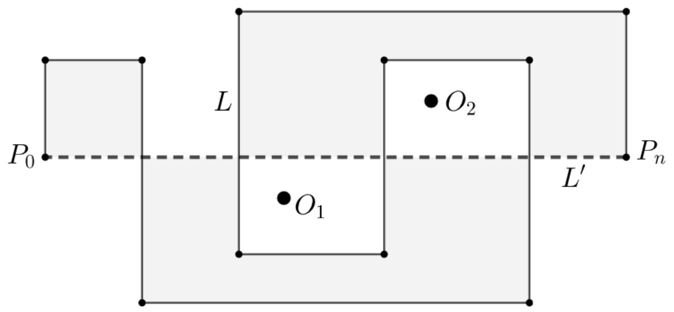


Figure 1.8: Topological consistency using the bounded face (grey region) of $L$ and $L'$ in the context of non-bounded $O_1$ and $O_2$.

The deformation (with fixed endpoints, remove only degree 2 vertices) of $L$ in Fig. 1.8 is illustrated in Fig. 1.9. In Fig. 1.9, keeping $P_0$ and $P_n$ fixed, only $P_1$, $P_2$, $P_3$ and $P_{n-1}$ of $L$ can be removed without passing over $O_1$ or $O_2$ in the non-bounded region.

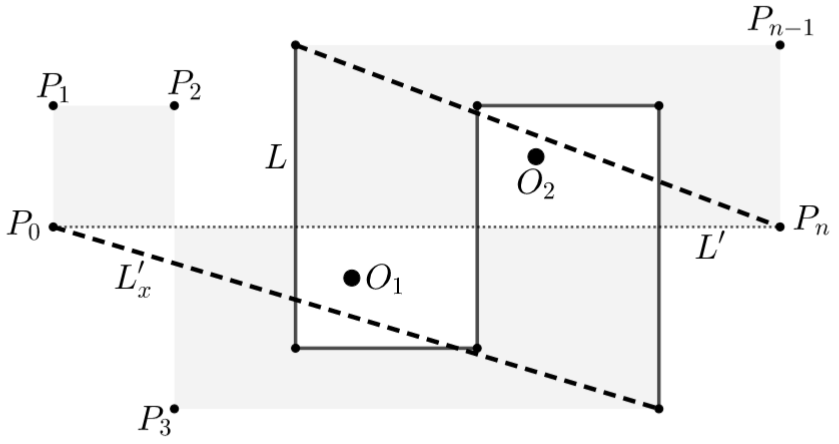


Figure 1.9: Homotopic deformation of $L$ in Fig. 1.8 towards $L'$ constrained by non-bounded $O_1$ or $O_2$. $L'_x$ is a temporary deformed edge of $L$ towards $L'$.

da Silva and Wu [2006] demonstrated the bounded face method by de Berg et al. [1998] is ill-defined for contextual linear features (handling planar linear objects as points). They also demonstrate the triangle inversion algorithm by Saalfeld [1999] can lead to self-intersection. The strategy of da Silva and Wu [2006] is to apply sidedness criterion to each sub-polyline and its collapsed segment. Let $D$ be a set of vertices of context linear features, $L$ is consistent to $L'$ if polygons formed by each sub-polyline $L_{i,j}$ and its segment $\overline{P_iP_j}$ contain no point in $D$. They test for sidedness using the parity (odd-even) rule. Their approach requires a vertex of a context object must be individually checked against each sub-polyline of $L$ for consistency. The restrictive algorithm by da Silva and Wu [2006] showed consistent sidedness of point and lines while avoiding intersections between linear features. The approach by da Silva and Wu [2006] in using vertices of neighbouring polylines as context constraints can be restrictive. In a subdivision (e.g., contours), vertices of neighbouring contours that may not exist after simplification are being used to restrict the simplification $L$.

Kulik et al. [2005] developed a $Min{-}\varepsilon$ ontology-driven simplification algorithm (DMin) modelled as a connected graph. Without a pre-specified fixed threshold ($Min{-}\varepsilon$ problem), DMin uses a geometric and semantic weight to remove vertices. Using a triangle-criterion, DMin avoids self-intersection and preserves non-planar topology by keeping track of intersections that are not vertices of the input polylines. The DMin algorithm by Kulik et al. [2005] depends on a connected graph from input polylines to maintain topological consistency. In a disconnected graph, DMin can introduce topological inconsistencies.

Estkowski and Mitchell [2001] also developed a graph structure for a planar subdivision ($S$ - non-crossing but can share endpoints) in the context points as planar constraints. Isolated feature points and vertices with degree one or greater than two are preserved in $S'$. They implemented a "Simple Detours" (SD) heuristic for chains in $S'$ by "untangling" it to remove intersections introduced by $\varepsilon$-feasible chains. Estkowski and Mitchell [2001] extended the SD algorithm to maintain the same homotopy type for points in a subdivision simplification. Their approach has not be demonstrated for arbitrary polylines with arbitrary context objects.

Abam et al. [2014] finds a polynomial solution for a simple polyline that is homotopic in the context of points as planar constraints. They introduce the concept of strongly homotopic where every shortcut in $L'$ is homotopic to the sub-polyline $L_{i,j}$. The homotopic algorithm by Abam et al. [2014] is based on Cabello et al. [2004] developed for simple lines in the context of points. The solution provide by Abam et al. [2014] except for $x$-monotone is not guaranteed to produce a simple simplification.

Funke et al. [2017] explores map simplification with a local topology constraint (points) using integer linear programming. Their method builds and maintains a constrained Delaunay triangulation of the subdivision and and all topology constraints. Vertices of degree 2 are removed one by one while maintaining the constrained triangulation for topological consistency. Funke et al. [2017] maintains local topological consistency using the bounded face technique by de Berg et al. [1998]. A polygon (possibly self-intersecting) created by a local shortcut and its sub-polyline does not contain any constraint point based on the even-odd-rule by Shimrat [1962] (see limitations of this test in Fig. 1.9).

## 1.5 Research Topic

Topology constraints for simple polylines or monotone ($x$ or $xy$) chains have been explored by but not limited to: Imai and Iri [1988], de Berg et al. [1998], Saalfeld [1999], Agarwal and Varadarajan [2000], Estkowski and Mitchell [2001], Cabello et al. [2004], Kulik et al. [2005], da Silva and Wu [2006], Dyken et al. [2009], Corcoran et al. [2011], Daneshpajouh and Ghodsi [2011], Abam et al. [2014], and Funke et al. [2017]. Most research literature that handle topological sidedness often develop constrained simplification algorithms using points as planar constraints [Imai and Iri, 1988; Agarwal and Varadarajan, 2000; Estkowski and Mitchell, 2001; Cabello et al., 2004; Abam et al., 2014; Funke et al., 2017].

For any arbitrary chain $L$, $L$ can be simple or complex. The utility of $L'$ is lost if errors are introduced as a result of simplification. If $L$ is simple, $L'$ is expected to be simple (and vice versa when complex). In a group ($S$) of arbitrary polylines, $L$ can have planar and non-planar intersections with other polylines. The simplified set $S'$ should reflect the topological properties of $S$. Representative spatial relations: proximity, intersect/disjoint and sidedness between $S$ and other planar objects should be consistent within some user defined parameters in $S'$. These constraints appear to make the line simplification problem intractable.

Estkowski and Mitchell [2001] showed that simplification of a subdivision (a set of non-crossing polylines except at endpoints) without introducing new vertices (Steiner points) is in fact difficult to solve, even approximately: a solution cannot be obtained in polynomial time within a factor $n^{1/5-\varepsilon}$ of an optimal solution, for any $\varepsilon > 0$ unless $P = NP$. Guibas et al. [1993] demonstrated that the problem of approximating a polygonal subdivision is $NP$-hard; it is $NP$-complete with no Steiner point [Estkowski, 1998].

There is limited research in simplification of arbitrary polylines in the context of arbitrary planar objects (points, polylines, and polygons). Furthermore, online simplification of spatiotemporal streams in the context of arbitrary planar objects is limited. The focus of this dissertation is development of geometric heuristics for the simplification of arbitrary polylines in the context of arbitrary planar objects as constraints.

## 1.6 Research Statement

Let $L$ be an arbitrary polyline and $S$ a set of arbitrary polylines with $L \in S$. Let $L'$ and $S'$ be the *restricted* simplification of $L$ and $S$ respectively. $L$ can be treated as spatially independent from its feature class $S$ (e.g., trajectories that overlap spatially but are separated in time). $L$ can also be spatially dependent on the neighbouring polylines in $S$ that constrain its shape, for example, the topology of contour lines (disjoint from neighbours) and road network (with planar or non-planar intersections). Let $C$ be a set of arbitrary planar objects (point, polyline, polygons) in the embedding plane of $L$ or $S$. Let $Q$ be a set of optional simplification constraints:

1. avoid introducing new self-intersections,
2. preserve planar and non-planar intersections,
3. preserve intersect and disjoint relations with objects in $C$,
4. preserve proximity to objects in $C$, and
5. preserve homotopy or sidedness in the context of $C$.

Given $\varepsilon > 0$ and $C$ as planar constraints, we consider the following restricted line simplification problems:

1. Compute $\varepsilon$-approximation of $L$ given $Q$ in the context of $C$;
2. Compute $\varepsilon$-approximation of $S$ where each $L$ in $S$ is spatially dependent given $Q$; and
3. Compute an online spatiotemporal $\varepsilon$-approximation of $L$ given $Q$ in the context of $C$

In this dissertation, contributions in chapters 2, 3, and 4 provide topological simplification of a polyline or group of polylines in the context of other objects. We extend heuristics developed in chapter 3 to handle trajectory streams in an online environment in chapter 4.

## 1.7 Research Objectives

The objectives of this research are addressed based on the simplification problems posed section 1.6.

1. Develop a conceptual framework for contextual line simplification with pluggable variants of the RDP error function ($\varepsilon$).
2. Develop algorithms and heuristics to preserve topological properties of polylines and its spatial relation to other planar objects.
3. Extend heuristics and algorithms for constrained line simplification to online trajectory streams.
4. Develop an application to extend the cartographic tools of an open-source or commercial GIS package.

The objectives and goals of this research provides an integration of varied constraints in contextual simplification of arbitrary polylines.

## 1.8 Overview of Each Chapter

**Chapter 1** starts with an introduction and a brief background of line simplification. It details related work, the research problem, and a set of objectives for this dissertation. In **Chapter 2**, we outline a conceptual and theoretical development of line simplification in the context of other planar constraints. The main contribution of this chapter is the development of a contextual model using the Douglas-Peucker algorithm with a prioritized context-based reversion to maintain topology and proximity relation to planar objects. Contextual constrained simplification is achieved by: (1) developing a linear characteristic and constrained simplification model that converges to the original polyline to resolve topological errors, (2) extending a directional relation operator/signature, and (3) extending the Binary Line Generalization Tree (BLG-Tree).

**Chapter 3** improves on the work done in Chapter 2 by developing a set of novel geometric heuristics for constrained line simplification. Contributions in this chapter include:

1. prevention of self-intersection in simple polylines,
2. preservation of planar self-intersection in complex polylines and between groups of polylines,
3. preservation of consistent sidedness between consecutive links in $L'$,
4. preservation of homotopy between a polyline and its simplification in the context other planar objects (points, lines and polygons),
5. maintenance of geometric (intersect/disjoint) relations between a polyline and other planar objects, and
6. maintenance of a minimum distance relation to other planar objects.

This chapter integrates topology (self-intersection, intersect, disjoint, homotopy) with proximity constraints to contextual planar objects. It also demonstrates constrained simplification of a planar subdivision.

**Chapter 4** develops an online contextual trajectory simplification under topological constraints. This chapter show a first online arbitrary trajectory simplification with topology and proximity constraints. A consistent simplification is achieved with the following contributions:

1. avoids self-intersection as a result of simplification,
2. preserves non-planar self-intersection between simplification units,
3. preserves intersect, disjoint, distance, and sidedness relation to arbitrary context geometries, and
4. provides external simplification of arbitrary trajectories in an online environment.

In **Chapter 5**, we implement geometric heuristics (developed in Chapter 3) as a *python Add-In* to extend the cartographic toolbox of ESRI ArcGIS. This is to make practical this research in industry and academic environments. **Chapter 6** concludes this research with contributions, limitations and recommendations.

# References

Abam, M. A., Daneshpajouh, S., Deleuran, L., Ehsani, S., and Ghodsi, M. (2014). Computing homotopic line simplification. *Computational Geometry*, 47(7):728–739.

Agarwal, P. K. and Varadarajan, K. R. (2000). Efficient algorithms for approximating polygonal chains. *Discrete & Computational Geometry*, 23(2):273–291.

Arge, L., Truelsen, J., and Yang, J. (2014). Simplifying massive planar subdivisions. In *2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 20–30. SIAM.

Buttenfield, B. (1985). Treatment of the cartographic line. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 22(2):1–26.

Cabello, S., Liu, Y., Mantler, A., and Snoeyink, J. (2004). Testing homotopy for paths in the plane. *Discrete & Computational Geometry*, 31(1):61–81.

Chan, W. and Chin, F. (1992). Approximation of polygonal curves with minimum number of line segments. In *International Symposium on Algorithms and Computation*, pages 378–387. Springer.

Corcoran, P., Mooney, P., and Winstanley, A. (2011). Planar and non-planar topologically consistent vector map simplification. *International Journal of Geographical Information Science*, 25(10):1659–1680.

Cromley, R. G. (1991). Hierarchical methods of line simplification. *Cartography and Geographic Information Systems*, 18(2):125–131.

da Silva, A. C. G. and Wu, S.-T. (2006). A robust strategy for handling linear features in topologically consistent polyline simplification. In *VIII Brazilian Symposium on Geoinformatics, 19-22 November, Campos do Jordão, São Paulo, Brazil*, pages 19–34.

Daneshpajouh, S. and Ghodsi, M. (2011). A heuristic homotopic path simplification algorithm. In *International Conference on Computational Science and Its Applications*, pages 132–140. Springer.

de Berg, M., van Kreveld, M., and Schirra, S. (1998). Topologically correct subdivision simplification using the bandwidth criterion. *Cartography and Geographic Information Systems*, 25(4):243–257.

Douglas, D. H. and Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122.

Dyken, C., Dæhlen, M., and Sevaldrud, T. (2009). Simultaneous curve simplification. *Journal of geographical systems*, 11(3):273–289.

Estkowski, R. (1998). No steiner point subdivision simplification is np-complete. *In Proceedings of the 10th Canadian Conference on Computational Geometry*.

Estkowski, R. and Mitchell, J. S. (2001). Simplifying a polygonal subdivision while keeping it simple. In *Proceedings of the seventeenth annual symposium on Computational geometry*, pages 40–49. ACM.

Funke, S., Mendel, T., Miller, A., Storandt, S., and Wiebe, M. (2017). Map simplification with topology constraints: Exactly and in practice. In *2017 Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 185–196. SIAM.

Goethem, A. v., Meulemans, W., Reimer, A., Haverkort, H., and Speckmann, B. (2013). Topologically safe curved schematisation. *The Cartographic Journal*, 50(3):276–285.

Goodrich, M. T. (1995). Efficient piecewise-linear function approximation using the uniform metric. *Discrete & Computational Geometry*, 14(4):445–462.

Gribov, A. and Bodansky, E. (2004). A new method of polyline approximation. *Lecture notes in computer science*, pages 504–511.

Guibas, L. J., Hershberger, J. E., Mitchell, J. S., and Snoeyink, J. S. (1993). Approximating polygons and subdivisions with minimum-link paths. *International Journal of Computational Geometry & Applications*, 3(04):383–415.

Guilbert, E. and Saux, E. (2008). Cartographic generalisation of lines based on a b-spline snake model. *International Journal of Geographical Information Science*, 22(8):847–870.

Heckbert, P. S. and Garland, M. (1997). Survey of polygonal surface simplification algorithms. Technical report, Carnegie-Mellon Univ. Pittsburgh PA., School of Computer Science.

Imai, H. and Iri, M. (1986). An optimal algorithm for approximating a piecewise linear function. *Journal of information processing*, 9(3):159–162.

Imai, H. and Iri, M. (1988). Polygonal approximations of a curve — formulations and algorithms. In Toussaint, G. T., editor, *Computational Morphology*, volume 6 of *Machine Intelligence and Pattern Recognition*, pages 71–86. North-Holland.

Jones, C. B. and Abraham, I. M. (1987). Line generalisation in a global cartographic database. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 24(3):32–45.

Keogh, E., Chu, S., Hart, D., and Pazzani, M. (2001). An online algorithm for segmenting time series. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 289–296. IEEE.

Kulik, L., Duckham, M., and Egenhofer, M. (2005). Ontology-driven map generalization. *Journal of Visual Languages & Computing*, 16(3):245–267.

Kurozumi, Y. and Davis, W. A. (1982). Polygonal approximation by the minimax method. *Computer Graphics and Image Processing*, 19(3):248–264.

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

Li, Z. and Openshaw, S. (1992). Algorithms for automated line generalization1 based on a natural principle of objective generalization. *International Journal of Geographical Information Systems*, 6(5):373–389.

Li, Z. and Openshaw, S. (1993). A natural principle for the objective generalization of digital maps. *Cartography and Geographic Information Systems*, 20(1):19–29.

McMaster, R. B. (1987). The geometric properties of numerical generalization. *Geographical Analysis*, 19(4):330–346.

Melkman, A. and O'Rourke, J. (1988). On polygonal chain approximation. In Toussaint, G. T., editor, *Computational Morphology*, volume 6 of *Machine Intelligence and Pattern Recognition*, pages 87–95. North-Holland.

Meratnia, N. and Rolf, A. (2004). Spatiotemporal compression techniques for moving point objects. In *International Conference on Extending Database Technology*, pages 765–782. Springer.

Muller, J. (1990). The removal of spatial conflicts in line generalization. *Cartography and Geographic Information Systems*, 17(2):141–149.

Ramer, U. (1972). An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing*, 1(3):244–256.

Reumann, K. and Witkam, A. (1974). Optimizing curve segmentation in computer graphics. In *Proceedings of the International Computing Symposium*, pages 467–472.

Saalfeld, A. (1999). Topologically consistent line simplification with the Douglas-Peucker algorithm. *Cartography and Geographic Information Science*, 26(1):7–18.

Saalfeld, A. (2000). Complexity and intractability: limitations to implementation in analytical cartography. *Cartography and Geographic Information Science*, 27(3):239–250.

Saux, E. (2003). B-spline functions and wavelets for cartographic line generalization. *Cartography and geographic information science*, 30(1):33–50.

Shi, W. and Cheung, C. (2006). Performance evaluation of line simplification algorithms for vector generalization. *The Cartographic Journal*, 43(1):27–44.

Shimrat, M. (1962). Algorithm 112: position of point relative to polygon. *Communications of the ACM*, 5(8):434.

Tienaah, T., Stefanakis, E., and Coleman, D. (2015). Contextual Douglas-Peucker simplification. *Geomatica*, 69(3):327–338.

Tienaah, T., Stefanakis, E., and Coleman, D. (2018a). Line simplification while keeping it simple or complex. In *Line Simplification Under Spatial Constraints*. Geographic Information Systems and Science - Research Lab, Geodesy and Geomatics Engineering, UNB.

Tienaah, T., Stefanakis, E., and Coleman, D. (2018b). Topologically consistent online trajectory simplification. In *Line Simplification Under Spatial Constraints*. Geographic Information Systems and Science - Research Lab, Geodesy and Geomatics Engineering, UNB.

Van Oosterom, P. (1991). The reactive-tree: A storage structure for a seamless, scaleless geographic database. In *AUTOCARTO Conference*, volume 6, pages 393–393. ASPRS.

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

Weibel, R. (1997). Generalization of spatial data: Principles and selected algorithms. *Algorithmic foundations of geographic information systems*, pages 99–152.

White, E. R. (1985). Assessment of line-generalization algorithms using characteristic points. *The American Cartographer*, 12(1):17–28.

# Chapter 2:

# Contextual Douglas-Peucker Simplification

## 2.1 Abstract

In this paper, we develop a constrained Douglas-Peucker algorithm using a polyline to be simplified and other geometries as contextual constraints. We develop a contextual model that incrementally rewinds to the original polyline with relevant characteristic vertices to resolve contextual conflicts. Constraints covered in this paper are topology and direction. Our implementation shows a consistent representation and a technique to accelerate multi-scale simplification of polylines.

## 2.2 Introduction

Line simplification is a fundamental process in cartographic generalization [Cromley, 1991; Weibel, 1997]. As an operator, simplification reduces redundancy or the level of detail required to represent a polyline at a desired scale. Line simplification is a well-covered topic with many algorithms in cartography, computational geometry, and computer graphics [Ramer, 1972; Douglas and Peucker, 1973; Reumann and Witkam, 1974; Opheim, 1982; Li and Openshaw, 1992; Normant and Tricot, 1993; Visvalingam and Whyatt, 1993; Fritsch and Lagrange, 1995; Balboa and López, 2000; Doihara et al., 2002; Qingsheng et al., 2002; Wu and Deng, 2003; Saux, 2003; Gribov and Bodansky, 2004; Kulik et al., 2005; Guilbert and Saux, 2008; Abam et al., 2010; Daneshpajouh and Ghodsi, 2011; Wang et al., 2012; Liu et al., 2012; Raposo, 2013; Abam et al., 2014]. One of the most popular algorithms in line simplification is the Douglas-Peucker (DP) algorithm [Douglas and Peucker, 1973]. The DP algorithm, like many other techniques of simplification, has an implicit scope to reduce the set of vertices used to represent a polyline based on some criteria (distance offset, error bound, direction/angular deflection, area, and other measures). The reduction technique is simple but obtaining a consistent or meaningful output is a complicated problem. The issues of consistent simplification arise from the fact that real world geometric properties are often constrained by neighbouring objects within a given space. Simplification of geometric properties without context may change the meaning of such properties (semantic representation).

In recent times, there is a greater need to perform simplification due to multi-scale representation of online interactive maps, high spatial and temporal resolution of data collected using location enabled devices - equipped with Global Positioning System (GPS). Evidence of the need for line simplification is observable in widely-used web and desktop mapping libraries or application [ArcGIS-10.2, 2014; Leaflet, 2014; Openlayers, 2014; PostGIS, 2014]. As part of the Leaflet Geometry module, *L.LineUtil* (http://bit.ly/1zdZ3nD) provides utility functions for line simplification using the DP-Algorithm to improve vector map rendering. OpenLayers geometry library *ol.geom.flat.simplify.douglasPeucker* (*http://bit.ly/1vLJvaq*) is based on the a library extracted from Leaflet. PostGIS spatial database extension uses *ST_SimplifyPreserveTopology* (*http://bit.ly/1vhwBCg*) to perform topology preserving simplification. ESRI ArcGIS-10.2 cartographic toolset provides *Sim-*

*plifyLine_cartography* for DP line simplification.

The main contribution of this paper is the development and testing of a contextual model of the Douglas-Peucker algorithm with a prioritized context-based reversion to maintain topology and direction relation in an embedded space. We achieve consistent simplification by: (1) developing a line characteristic and constrained simplification model (sections 2.3-2.3.2), (2) extending a directional relation operator/signature (section 2.3.2), (3) extending the Binary Line Generalization Tree (section 2.4).

## 2.3 Conceptual Definitions

A polyline (e.g., road centre line or coastline) is composed of points as vertices (e.g., location, intersection) and can be composed to form polygons (e.g., administrative regions, soil, forest and other thematic features as polygons). We use a modified subset of the OGC (Open Geospatial Consortium) simple feature specification of a LineString, Line, and LinearRing [OGC, 2014]. This paper defines a polyline (or often referred to as a "line") as a continuous chain of segments. A segment consists of a simple straight line connecting two points. A segment has a length property if the ends of the segment are separate in two-dimensional Euclidean space. A segment with zero length (coincident end points) can be generalized as a point (Fig. 2.1a). A polyline is simple if it does not self-intersect and is a ring if the endpoint of the last segment is the begin point of the first segment (Fig. 2.1e). A self-intersecting ring forms a complex ring (Fig. 2.1f).

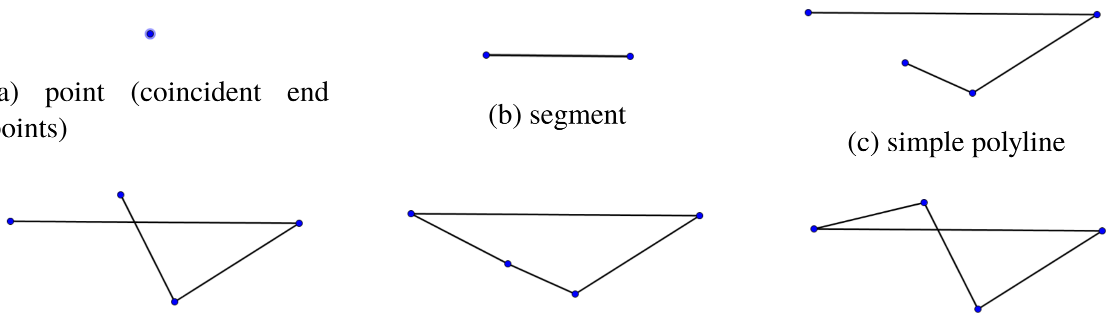
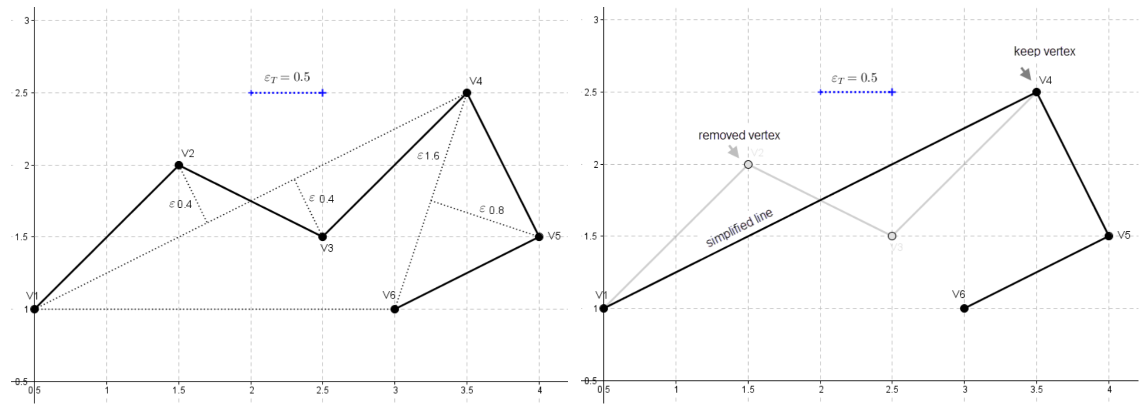

(a) point (coincident end points)
(b) segment
(c) simple polyline
(d) complex polyline
(e) simple ring
(f) complex ring

Figure 2.1: Line abstraction types.

Simplification of a line involves removal of redundant vertices based on a given criteria to reduce complexity in a dataset [Douglas and Peucker, 1973; Weibel, 1997]. Very often, linear simplification algorithms start with a polyline $L$ made up of two endpoints and an arbitrary set of vertices $V$. With a given criteria, $L$ is simplified into a polyline $L'$ by reducing the number of vertices of $V$ to $V'$, while keeping the ends of the polyline fixed. After simplification, $V'$ is either a proper subset of $V$ or equivalent to $V$; no new vertices are introduced nor vertices displaced. The classical criteria that guide vertex elimination are:

- minimize line distortion (no vertex of $L$ should be further away from $L'$ than a maximum error threshold - $\varepsilon_T$),
- minimize $V'$ (increase number of removed vertices or reduce data size), and
- minimize computational complexity (reduce cost of handling/rendering massive data).

In this paper, we perform constrained simplification of a polyline using the Douglas-Peucker algorithm. The simplification follows spatial and topological constraints that the geometric properties of a line and its relation to other neighbouring objects should be preserved [Mark, 1988; Stefanakis, 2012]. Given an ordered set of $N$ vertices ($V_{(1...N)}$) forming a polyline $L$, the Douglas-Peucker algorithm starts by marking the end points as “keep” ( $V_1'$ and $V_N'$). The algorithm finds the vertex $V_K$ from $V_2$ to $V_{(N-1)}$ with the maximum error offset ($\varepsilon_k$) from the line joining the end points ($V_1'$ and $V_N'$). If the $\varepsilon_k$ at $V_K$ is greater or equal to a pre-selected threshold $\varepsilon_T$ (where $\varepsilon_T > 0$), the vertex is marked as “keep” ($V_K'$). This process is repeated recursively by splitting the polyline at $V_K$ as two sub polylines $L_1$ ($V_{(1\ldots K)}$), and $L_2$ ($V_{(k\ldots N)}$) [Douglas and Peucker, 1973]. Figure 2.2 shows a graphical illustration of the algorithm. The recursion terminates if the maximum error offset is less than $\varepsilon_T$ or the polyline reduces to a line with only two vertices. The generalized line consists of all vertices marked as “keep”. Worst-time complexity for the Douglas-Peucker algorithm is $\mathcal{O}(n^2)$ and can be improved to $\mathcal{O}(n \log n)$ [Hershberger and Snoeyink, 1992].



(a) Original polyline

(b) Simplified polyline

Figure 2.2: Douglas-Peucker Simplification at 0.5units Maximum Error Threshold ($\varepsilon_T$= 0.5)

A polyline on map may exist with other spatial or thematic features. Neighbourhood geometries are context geometries that give some spatial or thematic meaning to a polyline on a map. We define a “neighbourhood geometry” as geometries intersecting the convex hull of vertices forming the polyline. The convex hull forms a polygon with a minimum set of vertices that envelope all the vertices of the polyline. A simplified line is a subset of all vertices of the original polyline, such a line is completely within or shares a boundary with the convex hull of the original polyline. Other geometries that intersect the convex hull may have disjoint, intersect or side relationship with the original polyline. A neighbouring geometry in relation to a polyline to be simplified can be a segment, polyline, point, or polygon. For example, Figure 2.3 illustrates the concept of neighbours with respect to convex hull (A).

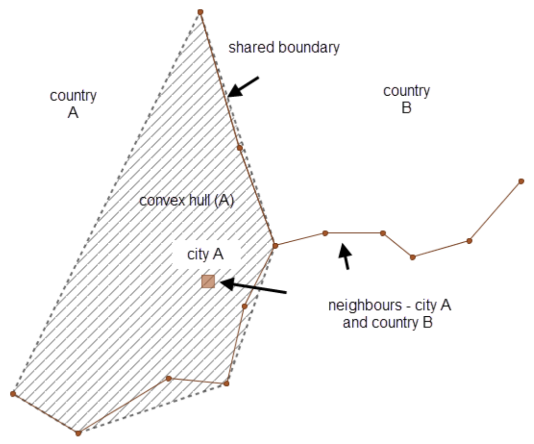
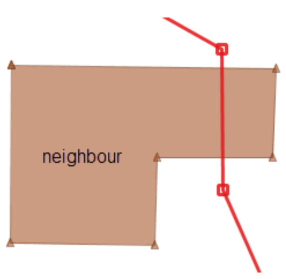


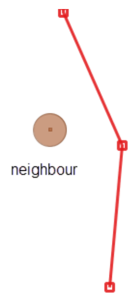
Figure 2.3: Neighbour geometries

### 2.3.1 Topological Relation

Polylines have internal (simple or self-intersection) or external topological relationships with other neighbouring geometries (intersect, disjoint, left, right, top, or down). During Douglas-Peucker simplification, the original polyline deforms into a new geometry with a smaller set of vertices; topological relations with other geometries may be violated if these geometries are found within the convex hull of the set of vertices forming the polyline [Saalfeld, 1999; Bertolotto and Zhou, 2007; Daneshpajouh and Ghodsi, 2011; Stefanakis, 2012].Our contextual model follows a strict geometric topological rule: the topological signature of the original and simplified polyline should be the same. Geometric relations are computed using the Dimensionally Extended nine-Intersection Model (DE-9IM) [Egenhofer and Franzosa, 1991; Clementini et al., 1993]. For example, a simplified road going through a park (a region or a polygon) is valid if the original geometry shares the same relation; else, it is an inconsistent representation (see intersect/disjoint relation in Fig 2.4).



(a) Intersection



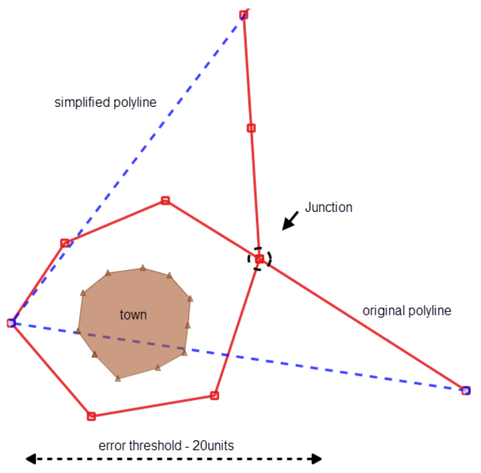
(b) Disjoint

Figure 2.4: Intersect and disjoint geometric relation

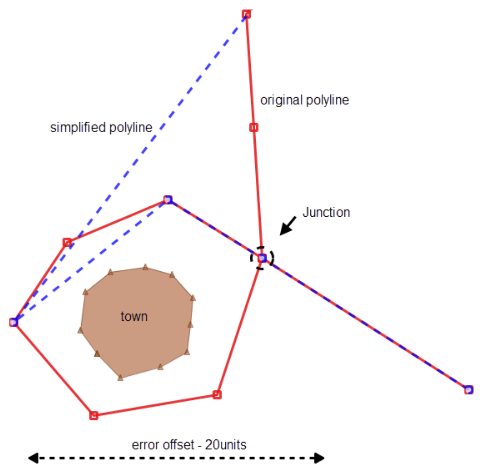
There have been various attempts to avoid self-intersection introduced as a result of DP simplification [Saalfeld, 1999; Mantler and Snoeyink, 2000; Estkowski and Mitchell, 2001; Bertolotto and Zhou, 2007; Shi and Charlton, 2013]. Real-world linear geometries sometimes self-intersect. To the best of our knowledge, there has been little research focus to maintain self-intersection as a constraint during DP-simplification. A typical example is a road geometry that goes around a town, roundabout or an obstacle and self-intersects at a junction. Figures 2.5a and 2.5b show DP simplification using Java Topology Suite (JTS), version 1.8.0 [Vivid-Solutions, 2014]. JTS is a library for 2D spatial predicate functions and spatial operators; it is one of the core and widely-used libraries in open source GIS (C++ port is used in PostGIS, GDAL/OGR, MapServer, QGIS, Shapely - Python).

### 2.3.2 Direction Relation

The direction relationship of a polyline describes its side relation with a neighbouring object. The simplified geometry must conform to the same quadrant relation as the original to avoid side (left, right, up, down, diagonal, and interior) conflicts. We extend the work of Theodoridis et al. [1998]. Figure 2.6a illustrates a nine cell direction relation model. It describes a quadrant based relationship between a line to be simplified and the object within its planar space. By registering the quadrants in which a line intersects relative to a neighbour, we create a directional signature for which the simplified version of the line must have to be topologically consistent. In Figure 2.6a, the linear geometry has a



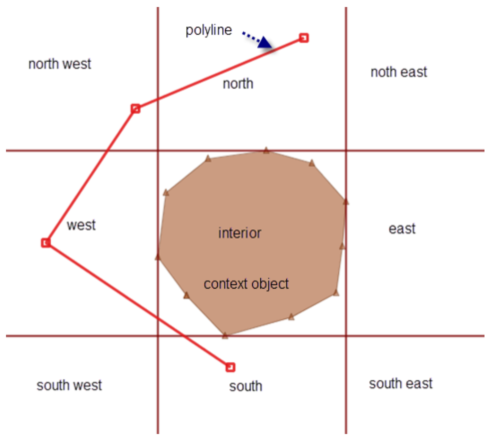
(a) JTS DP simplification



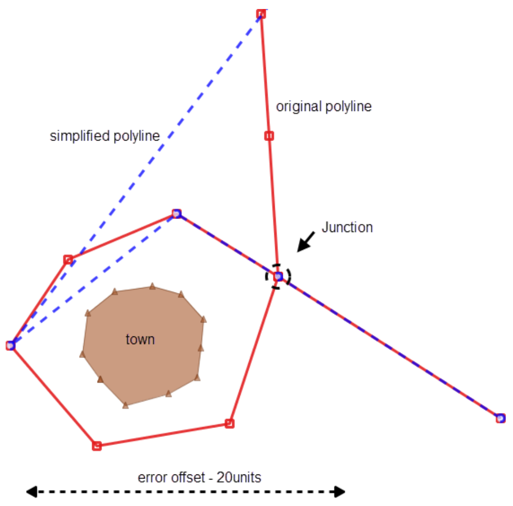
(b) JTS topology preserving DP simplification

Figure 2.5: A complex polyline with self-intersection. Original self-intersection not preserved (junction).

north, north-west, west, interior and south relation with respect to the polygon (neighbour) that intersects the convex hull of its vertices. The quadrants are defined by the minimum bounding box of a neighbouring object within the convex hull of the line to be simplified. Figure 2.6b shows a direction constraint violation; the original polyline registers a south relation with respect to the town, whereas the simplified geometry excludes this south relation.



(a) Nine cell side relation model



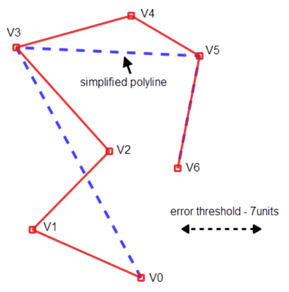
(b) Violation of side constraint (south)

Figure 2.6: Direction relation model (a) and DP simplification (b)

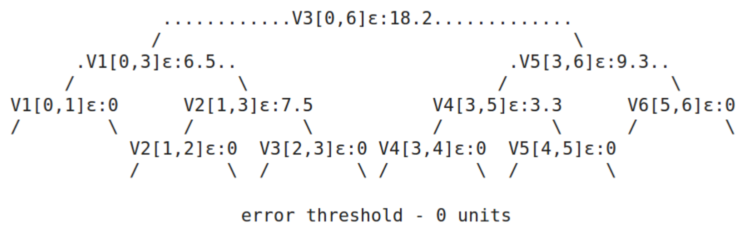
## 2.4 Contextual Rewind Model

The idea of building hierarchical structures from polylines started in the early 1980s. Ballard [1981] published a hierarchical representation using a binary tree called the "Strip Tree". The strip tree structure is a direct consequence of using a special method for digitizing lines and retaining all intermediate steps, each strip (oriented minimum bounding rectangle) consist of vertices of a polyline at each node. Van Oosterom [1991] introduced the Binary Line Generalization-tree (BLG-tree) which is a binary tree structure of the Douglas-Peucker algorithm.

As part of his research, the senior author of this paper extends the Binary Line Generalization tree (BLG-tree) [Van Oosterom, 1991] by keeping a prioritized list of all the intermediate error offsets ($\varepsilon$) and a convex hull of vertices forming the sub-polyline at each stage of the polyline decomposition (DP-algorithm). The extended BLG-tree is a direct binary tree structure of the DP algorithm. Each node of the tree has a reference to the two end points of the sub-polyline. The idea learned from the BLG-tree is to pre-process the polyline using an error threshold of zero (0) to obtain a binary tree structure of the original polyline (see Figure 2.7b). The extra storage (a prioritized list and convex hull of vertices at each node) in this extended BLG-tree structure is required for a rewind to relevant characteristic vertices (minimum set) given contextual information.



(a) DP simplification



```
              ............V3[0,6]ε:18.2.............
             /                                       \
      .V1[0,3]ε:6.5..                         .V5[3,6]ε:9.3..
     /               \                       /               \
V1[0,1]ε:0      V2[1,3]ε:7.5           V4[3,5]ε:3.3      V6[5,6]ε:0
/        \      /          \           /          \      /        \
           V2[1,2]ε:0  V3[2,3]ε:0 V4[3,4]ε:0  V5[4,5]ε:0
           /        \  /        \ /        \  /        \

                     error threshold - 0 units
```

(b) BLG-tree - excluding $V_0$

Figure 2.7: Extended BLG-Tree

Keeping the endpoints of the original polyline fixed, a vertex with maximum error offset

($\varepsilon$) forms the top-level node. From Figure 2.7b, if the $\varepsilon$ value of the root node ($V3$) is less than the predetermined simplification error threshold ($\varepsilon_T$), the simplified set of vertices is a null set ($\varnothing$). Line simplification as applied in the DP algorithm involves a search for the minimum set of vertices to represent the original line at an error threshold. A DP simplification in the context of other planar objects requires taking into consideration the relationship between the line and other geometries. In instances where the relationship is violated by the simplified line, it is important to reinstate some of the vertices removed, hence a rewind towards the original line.

Figure 2.8 shows a DP contextual rewind model developed by the senior author of this paper. All geometries that may become neighbours are indexed using an R-Tree. The model starts with a polyline, using $\varepsilon_T$ as zero, we pre-process the polyline into a binary tree. With a pre-processed binary tree structure, the simplification at a given $\varepsilon_T$ is a binary search for farthest nodes with $\varepsilon \leq \varepsilon_T$ from the root node (depth-first search). A search path terminates if a node with $\varepsilon \leq \varepsilon_T$ is encountered. Using the convex hull at each node, we search the R-Tree for neighbours that intersect the convex hull of the polyline being simplified. Where a geometric relation or direction is violated, the next vertex from a prioritized list of error offsets is added to the simplified sub-polyline at that node. Conflict resolution terminates when the topology relation is resolved. In some highly constrained cases, the algorithm will “rewind” - or revert to the original geometry (simplification cannot proceed without violating a topology or direction relation of the original polyline. See the “rewind loop” in Figure 2.8).

Note that the binary tree structure is not balanced and its efficiency depends on the shape of the original polyline. In the worst case, the binary tree reduces to a link list of nodes. Furthermore, nodes down the tree may have higher $\varepsilon$ values than its parent node (See Figure 2.7b, $\varepsilon$ value of node $V2$ versus the parent $V1$) [Van der Poorten et al., 2002]. The pre-processed binary tree structure accelerates finding characteristic vertices from a global decomposition as applied in the DP-algorithm. The structure also allows multilevel

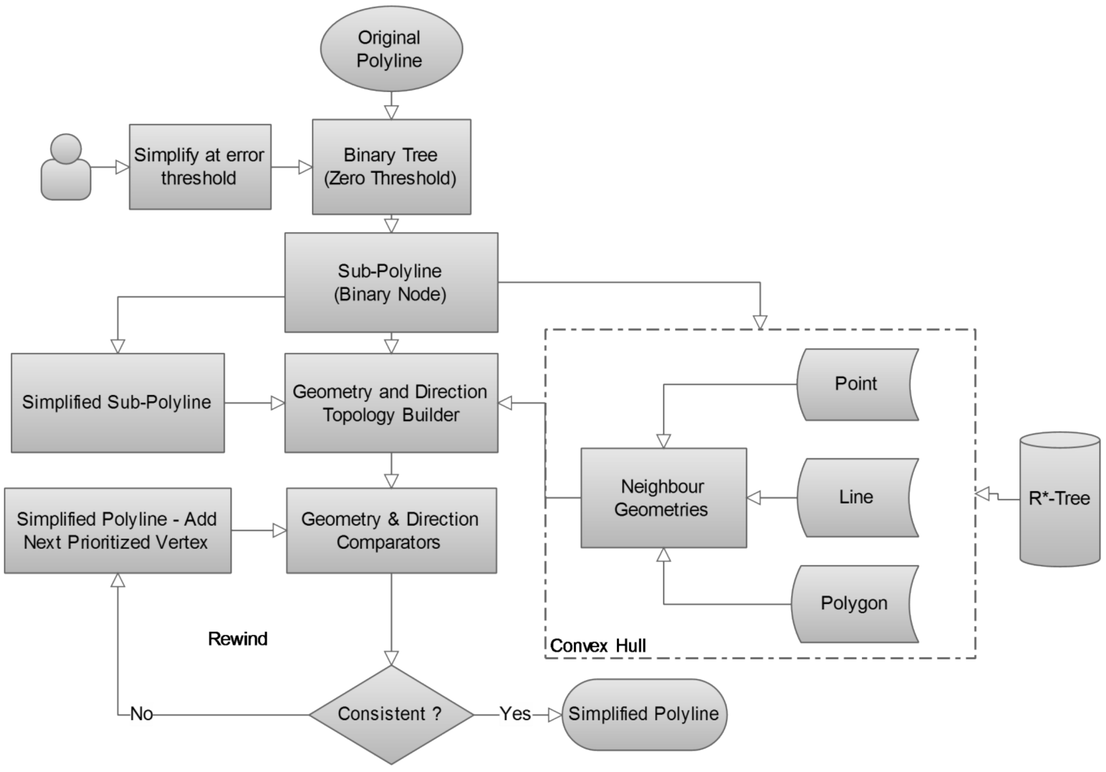
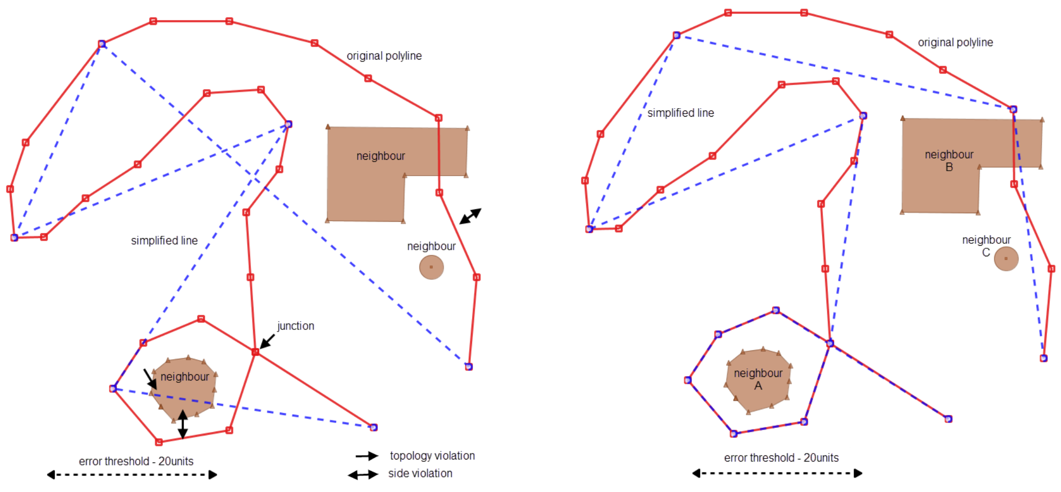

Original Polyline
Simplify at error threshold
Binary Tree (Zero Threshold)
Sub-Polyline (Binary Node)
Simplified Sub-Polyline
Geometry and Direction Topology Builder
Point
Neighbour Geometries
Line
R*-Tree
Simplified Polyline - Add Next Prioritized Vertex
Geometry & Direction Comparators
Polygon
Rewind
Convex Hull
No
Consistent ?
Yes
Simplified Polyline

Figure 2.8: Contextual rewind model

simplification since the entire line is represented once in the binary tree.

Figures 2.9a and 2.9b respectively show an unconstrained and constrained DP simplification; self-intersections resulting from DP simplification are not addressed in this example. Self-intersection consistency has been explored by many authors in cartography and computational geometry [Saalfeld, 1999; Mantler and Snoeyink, 2000; Wu and Marquez, 2003; Wu et al., 2004; Bertolotto and Zhou, 2007; Corcoran et al., 2011; Li et al., 2013]. In Figure 2.9a both topology and direction relation are violated because the planar geometries as neighbours are not considered during simplification. Figure 2.9b illustrates a consistent (preserves topology and direction) simplified line with contextual neighbours as constraints.

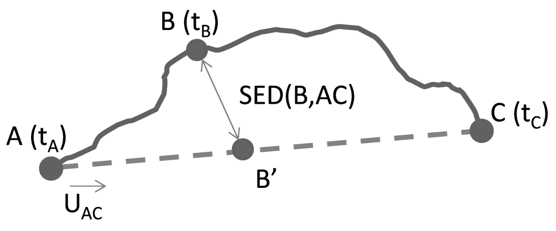


(a) (b)

Figure 2.9: Unconstrained (a) and constrained (b) DP Simplification

## 2.5 Trajectory Extension

The input data for the DP algorithm is an ordered set of points. A trajectory is a time ordered set of positions of a moving object. Essentially, it is a chain of segments joining points with a temporal (time) component. The DP algorithm by itself is not able to consider the temporal dimension of a trajectory. This is achieved by introducing the notion of the Synchronous Euclidean Distance (SED) [Meratnia and Rolf, 2004]. In a trajectory, each point $P_i$ is assigned a temporal stamp ($t_i$), which indicates the time a moving object crossed $P_i$. $A$, $B$, and $C$ are three spatiotemporal locations recorded for a trajectory T, with $t_A < t_B < t_C$ (Figure 2.10a). The SED for the point $B$ is equal to the Euclidean distance $BB'$, where the location $B'$ is the position $B$ on the simplified line (dash line) $AC$ with respect to the velocity vector $U_{AC}$ (Figure 2.10b). In other words, the error offset ($\varepsilon$) of $B$ is the distance from point $B$ to $B'$, where $B'$ is the spatiotemporal trace of $B$ on the straight-line approximation $AC$ at time $t_B$. The notion of SED only changes how we compute the error offset ($\varepsilon$) for the DP algorithm.



(a)

$$U_{AC} = \frac{AC}{t_C - t_A}$$

$$AB' = U_{AC}(t_B - t_A)$$

(b)

Figure 2.10: The Synchronous Euclidean Distance.

## 2.6 Evaluation

The DP and SED algorithms are implemented using *Node.js*, a JavaScript platform on top of *Google*'s *V8 JavaScript* engine. Vessel Trajectory Data is obtained from Marine Traffic AIS (*www.marinetraffic.com*) as comma-delimited files (CSV) with the following fields: *vessel id*, *latitude*, *longitude*, *time*, *speed*, *course* and other attributes. The project area is the Aegean Sea, between the mainlands of Greece and Turkey. This site provides a suitable set of islands that act as contextual constraints in the same planar space as the trajectories. Figure 2.11 shows a set of one hundred trajectories in the Aegean.

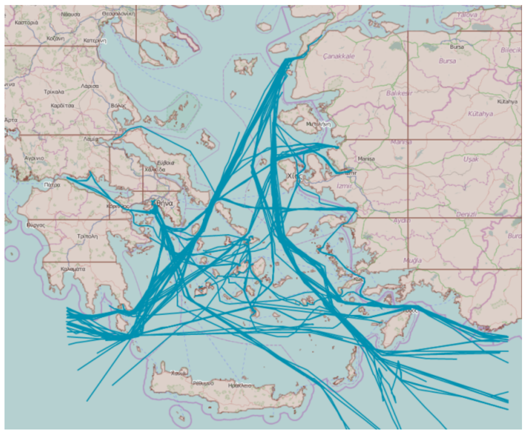


Figure 2.11: Study Area

A sample trajectory with $1,243$ vertices is as shown in Figure 2.12, each dot represents a vertex with spatiotemporal data.

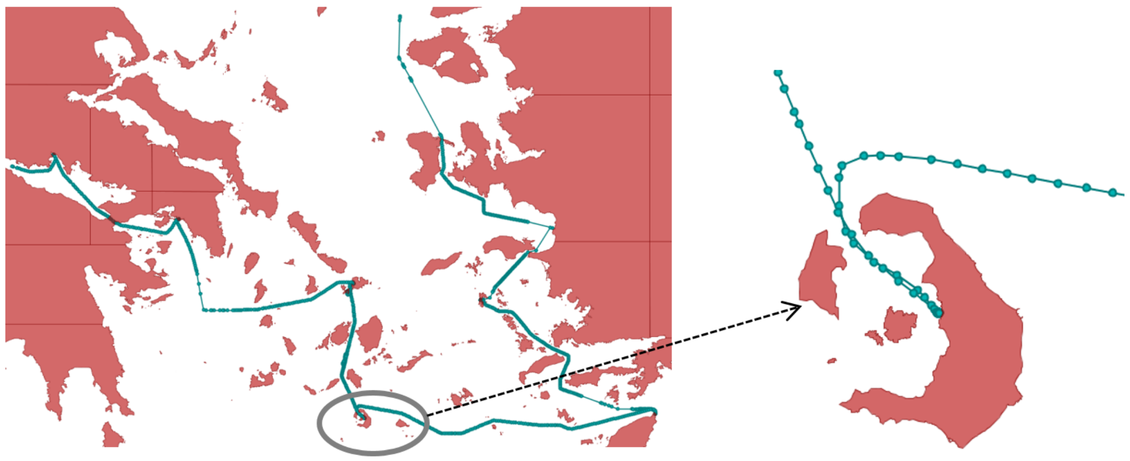


Figure 2.12: Sample Trajectory

Using the same trajectory illustrated in Figure 2.12, a constrained SED simplification at $5km$ distance threshold is illustrated in Figure 2.13. A comparative simplification without contextual constraints is illustrated in Figure 2.14. In Figure 2.14, the disjoint relation as shown in Figure 2.12 is violated. Figure 2.15a illustrates a section of an original trajectory and Figure 2.15b shows both constrained and unconstrained SED simplification. It can be can be observed in Figure 2.15b that the simplified line at $5km$ threshold violates both topology and direction if not constrained.

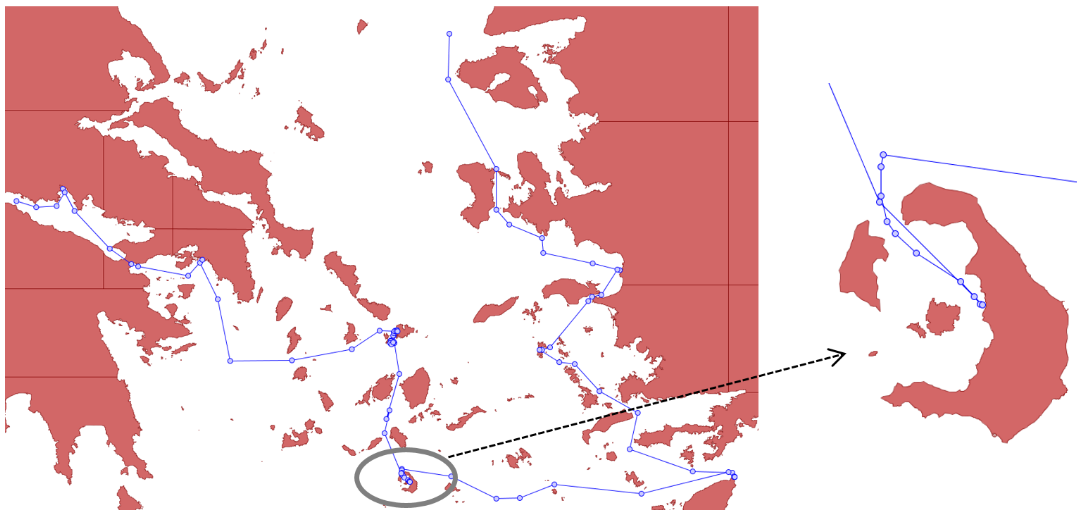


Figure 2.13: Constrained SED Simplification at $5km$ distance threshold

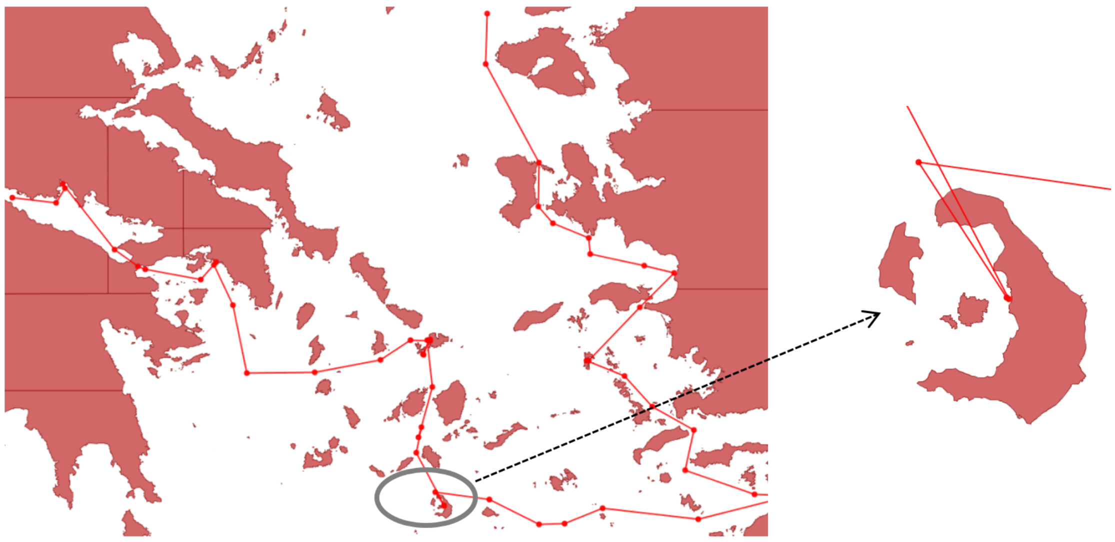
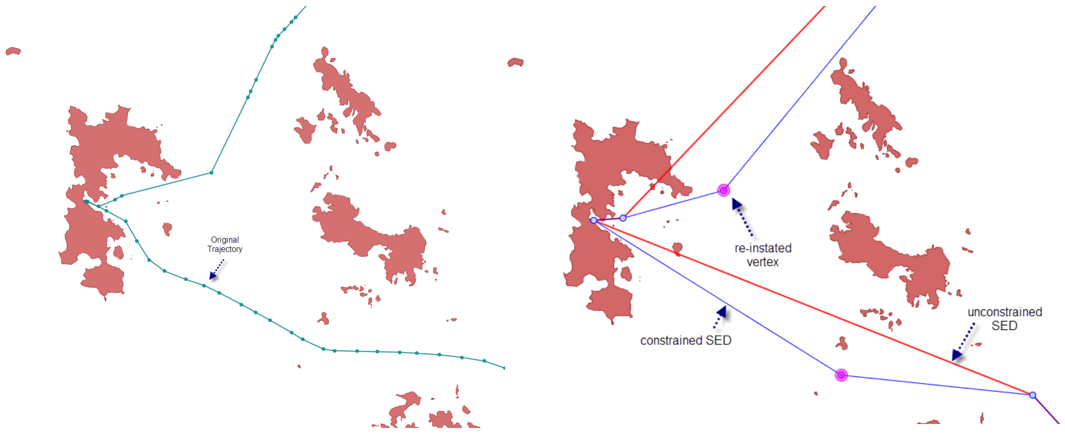


Figure 2.14: Unconstrained SED Simplification at $5km$ distance threshold



(a) (b)

Figure 2.15: Original Trajectory (a), Constrained and Unconstrained SED Simplification (b)

To demonstrate the effectiveness of context based simplification, we perform a post empirical evaluation of constrained and unconstrained SED simplification at SED thresholds starting from $5km$ to $50km$. Figure 2.16 illustrates percentage of vertices removed (PVR) versus SED thresholds. PVR is computed as a ratio of vertices removed as a result of simplification to the total number of vertices in the original trajectory. Figure 2.16 shows that, compression ratio decreases with increasing offset SED distance in constrained simplification as compared to a gradual increase in unconstrained simplification.

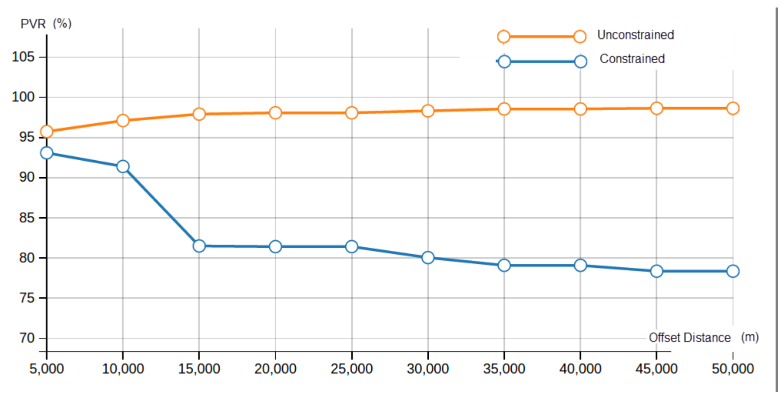


Figure 2.16: Percentage of redundant vertices (PVR versus $\varepsilon$)

The context based simplification presented in this paper reinstates removed vertices to resolve topology and direction conflicts; as a result, the simplified line is less displaced from its original. In Figure 2.17, we show a total area of polygonal displacement (TAPD) [McMaster, 1987]. TAPD is the sum of all displacement polygons standardized by the length of the original line.

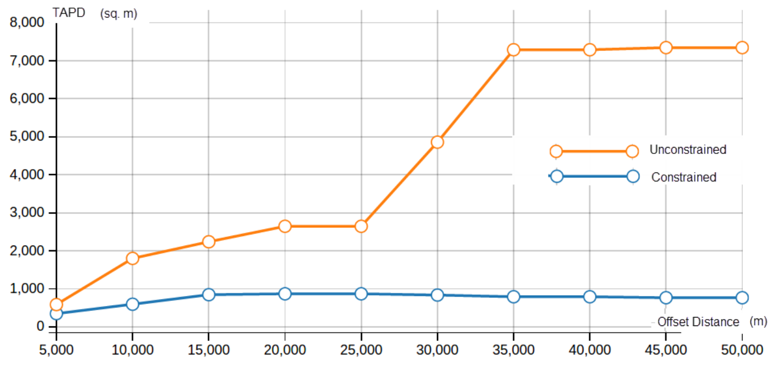


Figure 2.17: Total Area of Polygonal Displacement (TAPD versus $\varepsilon$)

## 2.7 Conclusion

In this paper, we have developed a contextual rewind model by extending the binary line generalization tree (BLG-tree) based on the Douglas-Peucker algorithm. We incorporate a stepwise topology and direction constraint at each binary node to maintain a consistent simplification. Constraint violations are resolved by reinstating a set of prioritized vertices to resolve conflicts. The binary tree structure requires storage of all intermediate steps during pre-processing (performed once at an error threshold of zero). The storage penalty of the binary tree structure offsets the cost of re-computing error offsets at different representations of the same polyline in a multi-scale multi-representation environment. Future work will focus on localization of static and moving space constraints and other empirical evaluation metrics.

# References

Abam, M. A., Daneshpajouh, S., Deleuran, L., Ehsani, S., and Ghodsi, M. (2014). Computing homotopic line simplification. *Computational Geometry*, 47(7):728–739.

Abam, M. A., De Berg, M., Hachenberger, P., and Zarei, A. (2010). Streaming algorithms for line simplification. *Discrete & Computational Geometry*, 43(3):497–515.

ArcGIS-10.2 (2014). Cartography toolbox - simplifyline_cartography.

Balboa, J. L. G. and López, F. J. A. (2000). Frequency filtering of linear features by means of wavelets. a method and an example. *The Cartographic Journal*, 37(1):39–49.

Ballard, D. H. (1981). Strip trees: A hierarchical representation for curves. *Communications of the ACM*, 24(5):310–321.

Bertolotto, M. and Zhou, M. (2007). Efficient and consistent line simplification for web mapping. *International Journal of Web Engineering and Technology*, 3(2):139–156.

Clementini, E., Di Felice, P., and Van Oosterom, P. (1993). A small set of formal topological relationships suitable for end-user interaction. In *International Symposium on Spatial Databases*, pages 277–295. Springer.

Corcoran, P., Mooney, P., and Winstanley, A. (2011). Planar and non-planar topologically consistent vector map simplification. *International Journal of Geographical Information Science*, 25(10):1659–1680.

Cromley, R. G. (1991). Hierarchical methods of line simplification. *Cartography and Geographic Information Systems*, 18(2):125–131.

Daneshpajouh, S. and Ghodsi, M. (2011). A heuristic homotopic path simplification algorithm. In *International Conference on Computational Science and Its Applications*, pages 132–140. Springer.

Doihara, T., Wang, P., and Lu, W. (2002). An adaptive lattice model and its application to map simplification. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, 34(4):259–262.

Douglas, D. H. and Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122.

Egenhofer, M. J. and Franzosa, R. D. (1991). Point-set topological spatial relations. *International Journal of Geographical Information System*, 5(2):161–174.

Estkowski, R. and Mitchell, J. S. (2001). Simplifying a polygonal subdivision while keeping it simple. In *Proceedings of the seventeenth annual symposium on Computational geometry*, pages 40–49. ACM.

Fritsch, E. and Lagrange, J. P. (1995). Spectral representations of linear features for generalisation. In *International Conference on Spatial Information Theory*, pages 157–171. Springer.

Gribov, A. and Bodansky, E. (2004). A new method of polyline approximation. *Lecture notes in computer science*, pages 504–511.

Guilbert, E. and Saux, E. (2008). Cartographic generalisation of lines based on a b-spline snake model. *International Journal of Geographical Information Science*, 22(8):847–870.

Hershberger, J. and Snoeyink, J. (1992). Speeding up the Douglas-Peucker line-simplification algorithm. Technical report, Vancouver, BC, Canada, Canada.

Kulik, L., Duckham, M., and Egenhofer, M. (2005). Ontology-driven map generalization. *Journal of Visual Languages & Computing*, 16(3):245–267.

Leaflet (2014). *L.LineUtil.simplify*.

Li, L., Wang, Q., Zhang, X., and Wang, H. (2013). An algorithm for fast topological consistent simplification of face features. *Journal of Computational Information Systems*, 9(2):791–803.

Li, Z. and Openshaw, S. (1992). Algorithms for automated line generalization1 based on a natural principle of objective generalization. *International Journal of Geographical Information Systems*, 6(5):373–389.

Liu, G., Iwai, M., and Sezaki, K. (2012). A method for online trajectory simplification by enclosed area metric. *ICMU'12*.

Mantler, A. and Snoeyink, J. (2000). Safe sets for line simplification. In *10th Annual Fall Workshop on Computational Geometry*.

Mark, D. M. (1988). Conceptual basis for geographic line generalization. *Proc. 9th International Symposium on Computer-Assisted Cartography*, pages 68–77.

McMaster, R. B. (1987). The geometric properties of numerical generalization. *Geographical Analysis*, 19(4):330–346.

Meratnia, N. and Rolf, A. (2004). Spatiotemporal compression techniques for moving point objects. In *International Conference on Extending Database Technology*, pages 765–782. Springer.

Normant, F. and Tricot, C. (1993). Fractal simplification of lines using convex hulls. *Geographical Analysis*, 25(2):118–129.

OGC (2014). Simple feature access-part 1:common architecture - version 1.2.1.

Openlayers (2014). *ol.geom.flat.simplify.douglasPeucker*.

Opheim, H. (1982). Fast data reduction of a digitized curve. *Geo-processing*, 2:33–40.

PostGIS (2014). *ST_SimplifyPreserveTopology*.

Qingsheng, G., Brandenberger, C., and Hurni, L. (2002). A progressive line simplification algorithm. *Geo-spatial Information Science*, 5(3):41–45.

Ramer, U. (1972). An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing*, 1(3):244–256.

Raposo, P. (2013). Scale-specific automated line simplification by vertex clustering on a hexagonal tessellation. *Cartography and Geographic Information Science*, 40(5):427–443.

Reumann, K. and Witkam, A. (1974). Optimizing curve segmentation in computer graphics. In *Proceedings of the International Computing Symposium*, pages 467–472.

Saalfeld, A. (1999). Topologically consistent line simplification with the Douglas-Peucker algorithm. *Cartography and Geographic Information Science*, 26(1):7–18.

Saux, E. (2003). B-spline functions and wavelets for cartographic line generalization. *Cartography and geographic information science*, 30(1):33–50.

Shi, S. and Charlton, M. (2013). A new approach and procedure for generalising vector-based maps of real-world features. *GIScience & remote sensing*, 50(4):473–482.

Stefanakis, E. (2012). Trajectory generalization under space constraints. In *Proceedings of the 7 th Intl. Conf. on Geographic Information Science (GIScience 2012). Columbus, Ohio*.

Theodoridis, Y., Papadias, D., Stefanakis, E., and Sellis, T. (1998). Direction relations and two-dimensional range queries: optimisation techniques. *Data & Knowledge Engineering*, 27(3):313–336.

Van der Poorten, P., Zhou, S., and Jones, C. B. (2002). Topologically-consistent map generalisation procedures and multi-scale spatial databases. In *International Conference on Geographic Information Science*, pages 209–227. Springer.

Van Oosterom, P. (1991). The reactive-tree: A storage structure for a seamless, scaleless geographic database. In *AUTOCARTO Conference*, volume 6, pages 393–393. ASPRS.

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

Vivid-Solutions (2014). *JTS topology suite*.

Wang, L., Guo, Y., Chen, C., and Yan, Y. (2012). A spatio-temporal data compression algorithm. In *Multimedia Information Networking and Security (MINES), 2012 Fourth International Conference on*, pages 421–424. IEEE.

Weibel, R. (1997). Generalization of spatial data: Principles and selected algorithms. *Algorithmic foundations of geographic information systems*, pages 99–152.

Wu, F. and Deng, H.-y. (2003). Using genetic algorithms for solving problems in automated line simplification. *Acta Geodaetica Et Cartographic Sinica*, 4:013.

Wu, S.-T., da Silva, A. C., and Márquez, M. R. (2004). The Douglas-Peucker algorithm: sufficiency conditions for non-self-intersections. *Journal of the Brazilian Computer Society*, 9(3):67–84.

Wu, S.-T. and Marquez, M. R. G. (2003). A non-self-intersection Douglas-Peucker algorithm. In *Computer Graphics and Image Processing, 2003. SIBGRAPI 2003. XVI Brazilian Symposium on*, pages 60–66. IEEE.

# Chapter 3:

# Line Simplification While Keeping it Simple or Complex

## 3.1 Abstract

We study the $Min{-}\#$ line simplification problem under the context of other geometries (point, polylines, polygons) as constraints. Unconstrained line simplification can lead to topological, proximity, and other spatial relational errors. Given an arbitrary polyline, a set context geometries, and $\varepsilon > 0$, we investigate a consistent simplification by observing these constraints: (i) preserve planar and non-planar intersections, (ii) avoid introducing new self-intersections as a result of simplification, (iii) preserve spatial relations to context geometries (disjoint and intersect), and (iv) preserve homotopy in simplification.

Our novel set of geometric heuristics are tested on real world data for correctness and compression effectiveness. In this paper we show that our constrained simplification is competitive in compression ratio compared to unconstrained.

## 3.2 Introduction

A polyline $L$ defined by $n$ line segments is an ordered connected chain of $n+1$ coordinates. Let the coordinates of $L$ be $P_0, P_i, ..., P_n$ where $P_0$ is $(x_0, y_0)$ and $P_n$ is $(x_n, y_n)$. The chain of $L$ is defined by segments: $\overline{P_0P_1}$, $\overline{P_1P_2}$, $\overline{P_2P_3}$, ..., $\overline{P_{n-1}P_n}$. The degree of a vertex is the number of segments incident on it [de Berg et al., 1998]. A vertex of degree one is an endpoint ($P_0$ or $P_n$, in a ring, $P_0 = P_n$); degree of two is a connecting or "interior" vertex that connects two segments (e.g., $P_1$ connects $\overline{P_0P_1}$, $\overline{P_1P_2}$). A vertex with degree greater than two is a planar "junction" (e.g., a road intersection). By keeping the endpoints ($P_0$ and $P_n$) fixed, the line simplification problem seeks the characteristic vertices of $L$ to form a new chain $L'$ with $m$ vertices such that the discarded coordinates from $L$ are at most $\varepsilon$ from $L'$, where $m < n$. The maximum $\varepsilon$ between $L$ and $L'$ is often termed as the simplification error [Douglas and Peucker, 1973; Imai and Iri, 1988; Hershberger and Snoeyink, 1992; Agarwal and Varadarajan, 2000].

Line simplification can be categorized into two forms: the $Min{-}\#$ and $Min{-}\varepsilon$ problem [Imai and Iri, 1988; Agarwal and Varadarajan, 2000]. The $Min{-}\varepsilon$ problem finds $L'$ with at most $K$ vertices that minimizes $\varepsilon$ over all approximations of $L$ that have $K$ vertices. The $Min{-}\#$ problem computes $L'$ that uses the smallest number of vertices among all $\varepsilon$-approximations of $L$ for $\varepsilon > 0$. The simplification problem can further be restricted or unrestricted. The restricted case requires the vertices of $L'$ to be a subset of $L$: $P_i = P_0, P_k, ..., P_j = Pn$ $(i < ... < j)$. The unrestricted case can introduce new vertices (Steiner points) in $L'$ that are not part of $L$. To approximate $L$ well, the coordinates of $L'$ are selected based on some criteria. In this paper, we consider the $Min{-}\#$ restricted problem and refer to the coordinates in $L'$ as the characteristic points of $L$ [White, 1985; McMaster, 1987b]. Irrespective of the simplification criteria, the sub-chain deformation between characteristic points to form a segment of $L'$ can lead to topological errors. In the context of other spatial objects, $L'$ can lead to spatial relational errors.

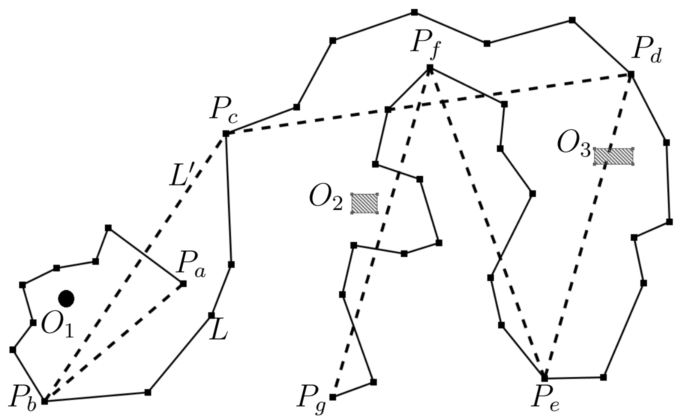


Figure 3.1: Approximation of $L$ (solid) as $L'$ (dashed) with context objects :$O_1$, $O_2$, $O_3$

Consider Fig. 3.1, starting from $P_a$ on the original polyline $L$, one will make a counter-clockwise turn at $P_b$ on the next consecutive chains of $L$ to $P_c$; on the simplification $L'$, $P_a$, $P_b$, and $P_c$ makes a clockwise turn at $P_b$. The topological representation in $L'$ inverts from $P_a$, $P_b$, to $P_c$. We refer to this simplification inversion in sub-polylines as the linear inversion problem. The segments $\overline{P_eP_f}$ and $\overline{P_fP_g}$ intersects $\overline{P_cP_d}$ in $L'$ (non planar self-intersection) which is topologically inconsistent with respect to $L$. In the context $O_1$, $\overline{P_aP_b}$ is on a different side of $O_1$ compared to $L$ from $P_a$ to $P_b$ - the sidedness of $O_1$ has changed between the original $L$ and its simplification $L'$ (inconsistent homotopy). $L'$ intersects $O_3$ whereas $L$ is disjoint. $L'$ can further be constrained by a minimum distance to other objects. Fig. 3.1 illustrates that simplification of $L$ in isolation can lead topological and spatial relational errors [Muller, 1990; Guibas et al., 1993; de Berg et al., 1998; Saalfeld, 1999; Estkowski and Mitchell, 2001]. The complexity and hardness of the problem have been explored by Saalfeld [2000], Estkowski and Mitchell [2001], and Guibas et al. [1993].

Linear features (e.g., rivers, roads, contours), boundaries of polygon geometries (e.g., parcel or county, coastline) or moving object trajectories (e.g., cars, animals, pedestrians, vessels) are important in cartography, GIS, Very Large Scale Integration (VLSI), image processing and computer graphics. The correct simple representation $L'$ is important for data storage, data transmission, and speed-up of graphic processing. Line simplification is a well studied problem; refer to Weibel [1997], Heckbert and Garland [1997], and Shi and Cheung [2006] for a comprehensive survey of simplification techniques. Other authors have studied and applied line simplification in cartography and GIS [Douglas and Peucker, 1973; Buttenfield, 1985; McMaster, 1987a; Cromley, 1991; Hershberger and Snoeyink, 1992; Li and Openshaw, 1992, 1993; Weibel, 1997], computer graphics, and computational geometry [Ramer, 1972; Suri, 1986; Melkman and O'Rourke, 1988; Imai and Iri, 1988; Chan and Chin, 1992; Hobby, 1993; Guibas et al., 1993; Goodrich, 1995; Miyaoku and Harada, 1998].

In this paper, we develop our simplification heuristics based on a popular line simplification algorithm independently developed by Ramer [1972] and Douglas and Peucker [1973] (RDP). The restricted $Min\!-\!\#$ problem is explored with the following scope and constraints:

1. Geometric objective: $L'$ should preserve disjoint or intersection relation with other objects in the planar space of $L$.

2. $Min\!-\!dist$ objective: $L'$ should maintain a certain minimum distance to context objects (preserve proximity relation if observed by $L$).

3. Homotopy objective: $L$ should be homotopic to $L'$ in the context of other geometries (point, polyline, or polygon).

4. Planar self-intersection: $L'$ should preserve planar self-intersection and avoid introducing new self-intersections as a result of simplification.

5. Non-planar self-intersection: $L'$ should preserve non-planar intersections of $L$ and with segments of other polylines.

6. Linear inversion: consecutive segment of $L'$ should not invert the topological relation in $L$.

The rest of the paper is organized into five sections. Section 3.3 discusses related work in constrained simplification. Section 3.4 introduces our line simplification algorithm. Constrained heuristics and algorithms are developed in section 3.5. Experimental evaluation of constrained simplification on real world data are presented in section 3.6. Section 3.7 concludes the this paper with contributions and future work.

## 3.3 Related Work

de Berg et al. [1998] developed a method of polygonal simplification of a subdivision based on the work of Imai and Iri [1988]. They modelled a polygonal chain ($L$) as a graph structure where its simplification ($L'$) are the shortest paths (allowed shortcuts) through the graph. The limitation of de Berg et al. [1998] is that, $L$ is required to be simple and $L'$ can intersect other chains of the subdivision. Their method maintained the sidedness of points $P$ relative to $L$ and $L'$ if $L$ is monotone [Corcoran et al., 2011]. Estkowski and Mitchell [2001] developed a graph structure for a set ($S$) of polygonal chains and feature points given $\varepsilon > 0$. Their implementation requires polylines in $S$ to be non-crossing but can share endpoints. Isolated feature points and vertices with degree one or greater than two are preserved in the simplification ($S'$). They implemented a "Simple Detours" (SD) heuristic for chains in $S'$ by "untangling" it to remove intersections introduced by $\varepsilon$-feasible chains. Estkowski and Mitchell [2001] extended the SD algorithm to maintain the same homotopy type in the context of points in a subdivision simplification. Kulik et al. [2005] developed a $Min{-}\varepsilon$ ontology-driven simplification algorithm (DMin) modelled as a connected graph. Without a pre-specified fixed threshold ($Min{-}\varepsilon$ problem), DMin uses a geometric and semantic weight to remove vertices. Using a triangle-criterion, DMin avoids self-intersection and preserves non-planar topology by keeping track of intersections that are not vertices of the input polylines.

The graph technique by de Berg et al. [1998] applies to simple and not arbitrary chains.

The SD algorithm by Estkowski and Mitchell [2001] require non-crossings polylines. The DMin algorithm by Kulik et al. [2005] depends on a connected graph from input polylines to maintain topological consistency. In a disconnected graph, DMin can introduce topological inconsistencies. Furthermore, arbitrary polylines such as a road network may not be simple (planar and non-planar crossings) limiting techniques for simple polylines in a subdivision simplification [de Berg et al., 1998; Estkowski and Mitchell, 2001]. The challenge of maintaining topological consistency between arbitrary polylines in the context of other objects have led some authors [Saalfeld, 1999; Mantler and Snoeyink, 2000; Wu and Marquez, 2003] to decompose a polyline into a series of safe simplification sub-polylines.

RDP decomposes a polyline into sub-polylines at characteristic vertices [Douglas and Peucker, 1973]. Saalfeld [1999] developed a triangle inversion property of vertices in the convex hull of each sub-polyline to avoid self-intersection. Mantler and Snoeyink [2000] used a point Voronoi diagram to decompose vertices of polyline into safe sets that can be simplified without introducing self-intersection. A safe set is an ordered set vertices that is monotonically increasing in some direction whose convex hull contains no other point. Wu and Marquez [2003] presented a variant of RDP by dividing a polyline into star-shaped subsets to prevent self-intersection. Algorithms by Mantler and Snoeyink [2000] and Wu and Marquez [2003] do not handle the relationship between sub-sets of other linear geometries that may have planar or non-planar intersections with polyline being simplified.

Saalfeld's [1999] algorithm applies to polyline simplification in the context of other objects as points. da Silva and Wu [2006] showed limitations of Saalfeld [1999] and demonstrates how to extend the method of de Berg et al. [1998] to handle context linear features by considering it as series of points. Corcoran et al. [2011] provided a unified understanding between de Berg et al. [1998]; Saalfeld [1999], and da Silva and Wu [2006]. Corcoran et al. [2011] further integrates the benefits of de Berg et al. [1998], Saalfeld [1999] and da Silva and Wu [2006] to provide a simplification algorithm that is less restrictive compared to da Silva and Wu [2006]. Corcoran et al. [2011] is limited with respect to self-crossing arbitrary polylines that can create non-bounded regions between $L'$ and $L$ (e.g., see Fig. 3.14). Also the point-in- polygon (even-odd rule) algorithm [Shimrat, 1962] as applied by de Berg et al. [1998], da Silva and Wu [2006], and Corcoran et al. [2011] can restricted by two non bounded objects, see Fig. 3.13. This method of homotopic simplification does not generalize well to arbitrary context objects such as polylines and polygons.

Abam et al. [2014] finds a polynomial solution for a simplified polyline that is homotopic to its input given a set of points as constraints. Abam et al. [2014] introduces the concept of strongly homotopic where every shortcut in $L'$ is homotopic to the sub-polyline in $L$. Funke et al. [2017] implements an integer linear programming and a heuristic for simplifying a planar subdivision in the context of points as planar constraints. Their method uses constrained Delaunay triangulation to maintain topology constraints. Vertices of degree 2 are removed one by one while maintaining the constrained triangulation for consistency. The algorithm by Abam et al. [2014] except $x$-monotone lines may result in self-intersection for simple polylines. Funke et al. [2017] maintains local topological consistency using the bounded face technique by de Berg et al. [1998]. A polygon (possibly self-intersecting) created by a local shortcut and its sub-polyline does not contain any constraint point based on the even-odd-rule by Shimrat [1962] (see limitations of this method in Fig. 3.13 and Fig. 3.14). Algorithms of Abam et al. [2014] and Funke et al. [2017] are based on simple polylines and have not been demonstrated to handle arbitrary polylines with linear and polygonal context constraints.

### 3.3.1 Research Contributions

Most algorithms found in literature are either restricted for a specific type of polyline (monotone or simple) and not arbitrary chains [de Berg et al., 1998; Saalfeld, 1999; Agarwal and Varadarajan, 2000; Estkowski and Mitchell, 2001; da Silva and Wu, 2006; Abam et al., 2014; Funke et al., 2017]. Attempts have been made to preserve planar and non-planar intersections in polylines [Estkowski, 1998; Kulik et al., 2005; Corcoran et al., 2012]. Techniques to avoid self-intersection as a result of simplification have also been explored [Saalfeld, 1999; Wu and Marquez, 2003; da Silva and Wu, 2006; Mantler and Snoeyink, 2000]. Contextual simplification has also been explored in research literature [de Berg et al., 1998; Abam et al., 2014; Funke et al., 2017]. There is lack of a unified constrained simplification algorithm that handles topological relations in an arbitrary polyline or a class of arbitrary polylines while preserving topology (homotopy, intersect/disjoint, self-intersection, linear inversion), and proximity constraints to arbitrary context objects (point, polylines, and polygons). We focus on a set novel geometric heuristics based on sub-polylines (simplification units) created by RDP to maintain topology and context spatial relations. Our contributions in this paper include:

1. prevention of self-intersection in simple polylines,
2. preservation of planar self-intersection in complex polylines and between groups of polylines,
3. prevention of linear inversion (mirroring in $L'$),
4. preservation of homotopy relation between a polyline and its simplification in the context other planar objects (points, lines and polygons),
5. maintenance of geometric (intersect/disjoint) relation between a polyline and other planar objects (points, lines and polygons),
6. maintenance of a minimum distance relation to planar objects.

Our strategy for resolving conflicts is similar in concept to Saalfeld [1999], Estkowski and Mitchell [2001]. Inconsistencies are resolved by introducing characteristic vertices of $L$ at $\varepsilon_k < \varepsilon$ - a further deformation of a sub-polyline using RDP (“untangling”).

## 3.4 Polyline Decomposition

In this paper, we use RDP to decompose a polyline ($L$) into sub-polylines. The subset and proximity property of RDP ensures a simplification is within a distance $\varepsilon$ of the original polyline and is a subset of the its vertices. The implementation is generic and adaptable to other algorithms by changing how the characteristic vertices are selected. RDP is used as a distance function (Hausdorff) for selecting the shape characteristics of a polyline. We describe RDP as a preliminary for our constrained simplification heuristics. Given a distance threshold $\varepsilon$ and an ordered set of $n + 1$ vertices $(P_0, P_1, P_i, ...P_n)$ forming a polyline $L$, RDP performs a recursive decomposition by keeping the endpoints ($P_0$ and $P_n$) fixed. It finds a splitting vertex $P_k$ by computing a distance offset of intermediate vertices $(P_1, ..., P_{n-1})$ from the generalized segment $\overline{P_0P_n}$. $P_k$ is the vertex with maximum of the minimum distances ($\varepsilon_k$) from each intermediate vertex $(P_1, ..., P_n - 1)$ to $\overline{P_0P_n}$. If $\varepsilon_k > \varepsilon$, split the input polyline at $P_k$ with sub-polylines $L_{0,k}$ $(P_0, P_1, P_i, ..., P_k)$ and $L_{k,n}$ $(P_k, P_{k+1}, P_j, ..., P_n)$. $L_{0,k}$ and $L_{k,n}$ become new inputs to the recursive decomposition, the recursion terminates if $\varepsilon_k \leq \varepsilon$ or $L_{i,j}$ is a segment ($j - i = 1$, where $j > i$). The complexity of RDP is $\mathcal{O}(n^2)$. Hershberger and Snoeyink [1992] observed the farthest vertex $P_k$ from the generalize line segment must be part of the convex hull of its vertices; they built data structures to maintain and access $P_k$ as a way of improving ($\mathcal{O}(n \log n)$) RDP for simple polylines; Hershberger and Snoeyink [1992] does not apply to arbitrary linear features.

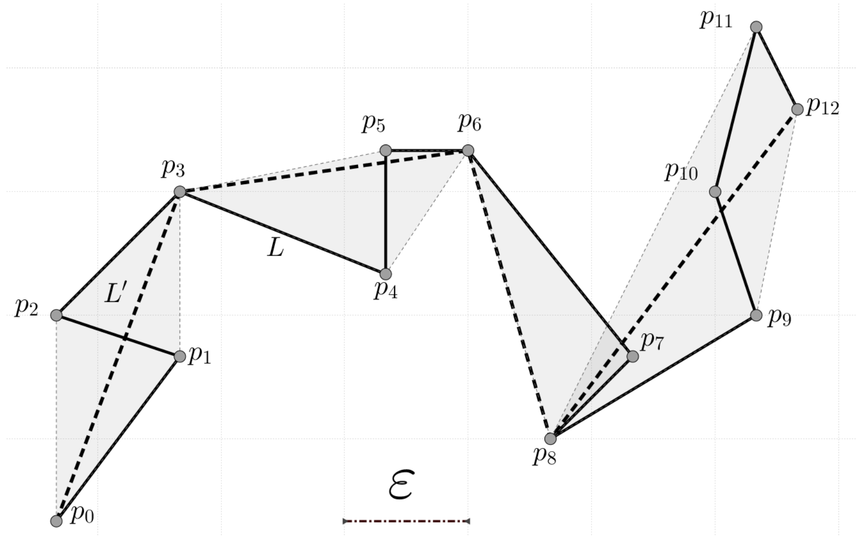


Figure 3.2: RDP Hulls

In Fig. 3.2, the solid polyline $L$ is the original input and the dashed $L'$ is a simplification of $L$ at $\varepsilon$. A shape deformation of $L$ is required to obtain $L'$, that is, a shape collapse of each sub-polyline $L_{k,q}$ into a line segment $\overline{P_kP_q}$. $k$ and $q$ are the vertex indices in $L$, where $q > k$. $L'$ is an ordered connected chain of collapsed characteristic segments. The shaded region in Fig. 3.2 illustrates the minimum enclosing area of each sub-polyline (convex hull of vertices). In this paper, we refer to the convex hull of a sub-polyline as a context hull ($CH$). Let $CH_{[k,q]}$ be the context hull of sub-polyline $L_{k,q}$, the context collapse of $CH_{[k,q]}$ is the line segment $\overline{P_kP_q}$ where $P_k$ and $P_q$ are the endpoints of $L_{k,q}$. Note that $P_k$ and $P_q$ may not be convex vertices of $CH_{[k,q]}$. Out of context collapse of $CH_{[k,q]}$ can lead to spatial and topological errors [de Berg et al., 1998; Saalfeld, 1999; Estkowski and Mitchell, 2001].

## 3.5 Constrained Line Simplification

To perform a constrained consistent RDP simplification, it is important to observe and resolve topological or spatial relational errors. The main focus of this paper is to consider topology and minimum distance relation as simplification objectives. We achieve and make an original research contribution by developing a series of algorithms using the context hull ($CH$) of each sub-polyline by observing the following constraints:

1. planar self-intersection - a shared vertex ($degree > 2$) within one or more polylines,
2. non-planar self-intersection between deformation units - $CH$ of one or other polylines,
3. intersect and disjoint geometric relation to other context objects (point, line, or polygon),
4. homotopy of simplified polyline in the context of other objects, and
5. minimum distance relation to other context objects.

$CH_{[i,j]}$ acts as a region of spatial influence for the generalized segment $\overline{P_iP_j}$. Since $\overline{P_iP_j}$ for $L_{i,j}$ cannot exist outside $CH_{[i,j]}$, errors that can occur as a result of simplification are related to inconsistent relations that exist within $CH_{[i,j]}$ or at a certain distance ($\delta$) from its boundary. We refer to geometries that intersect $CH_{[i,j]}$ or at $\delta$ from a its boundary as context neighbours. Context neighbours can be a points, lines, or polygons. Other context hulls of the same polyline or other polylines are treated as context neighbours. Splitting and merging of context hulls are central to algorithmic developments in this paper. We provide some properties and formal definitions.

- **Context hull (**$CH_{[i,j]}$**):** is the minimum enclosing area of a point set ($P_s$). Depending on the number of points in $P_s$, $CH_{[i,j]}$ can be represented geometrically as a line segment or polygon.

- **Deformable:** $CH_{[i,j]}$ is *deformable* if there is a selectable intermediate vertex $P_k$ at $\varepsilon_k$ where $i < k < j$ and $\varepsilon_k < \varepsilon$. Based RDP (see section 3.4), $P_k$ is the farthest vertex from $\overline{P_iP_j}$. $CH_{[i,j]}$ with $j - i = 1$ is not deformable.

- **Collapsible:** $CH_{[i,j]}$ is *collapsible* if it can be represented as $\overline{P_iP_j}$ without violating a pre-specified constraint.

- **Contiguous:** $CH_{[i,k]}$ and $CH_{[j,n]}$ are contiguous if $k = j$ or $i = n$. $CH_{[i,k]}$ and $CH_{[j,n]}$ are non-contiguous if $k \neq j$ and $i \neq n$.

- **Mergeable:** contiguous context hulls $CH_{[i,k]}$ and $CH_{[j,n]}$ are *mergeable* if $\varepsilon_q \leq \varepsilon$, where $P_q$ is the characteristic vertex from $\overline{P_iP_n}$, where $i < q < n$.

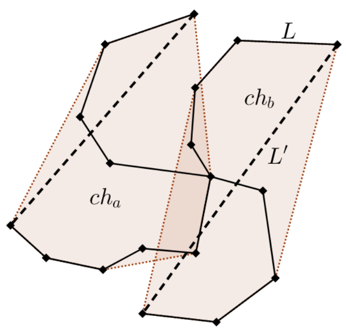
Some authors [de Berg et al., 1998; Saalfeld, 1999; Estkowski and Mitchell, 2001; da Silva and Wu, 2006; Tienaah et al., 2015] have considered strategies of further simplifying $L_{i,j}$ by introducing some of the original vertices to resolve topological errors. In this paper, we use this intuition to develop geometric heuristics to resolve topology and spatial relational errors. Consider a series of contiguous context hulls: $CH_{[0,3]}$, $CH_{[3,7]}$, $CH_{[7,9]}$ and $CH_{[7,13]}$. There is no one obvious strategy of collapsing the context hulls for an arbitrary polyline. A sequential collapse starting from $CH_{[0,3]}$ can lead topological errors in successive hulls; the reverse, stating from $\overline{P_7P_{13}}$ can also be inconsistent with preceding context hulls. A sequential collapse of each contiguous context hull as presented in Tienaah et al. [2015] can lead to unnecessary introduction of original vertices in later context hulls that may be inconsistent with preceding collapsed hulls. In this paper, we present an original set of heuristics that express the topological and spatial relationship between context hulls to precisely identify deformable and collapsible hulls. In a group of polylines (e.g., contour lines, road network), it is important the resulting simplification retains the spatial and topological characteristics of the input polylines.

### 3.5.1 $Min{-}\#$ polyline deformation heuristics

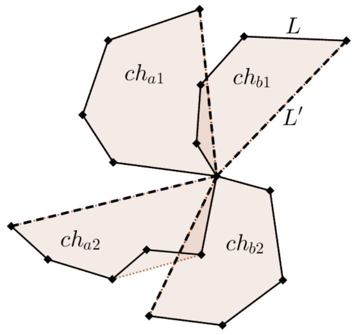
Let $P_i, P_{i+1}, P_k, ..., P_{j-1}, P_j$ be the coordinates of a sub-polyline $L_{i,j}$. The geometry of a context hull is a line segment if the intermediate vertices between $i$ and $j$ ($P_{i+1}, P_k, ..., P_{j-1}$) are collinear with $\overline{P_iP_j}$. The convex hull of $L_{i,j}$ with two coordinates ($j - i = 1$) is also represented as line segment. Context hulls with more than two coordinates with an areal extent are represented geometrically as polygons.

After RDP decomposition of $L$, $L_{i,j}$ can contain planar and non-planar self-intersection. A vertex $P_k$ shared by more than two line segments is further deformed at $P_k$ if preservation of planar self-intersection is required. Given a context hull $CH_{[i,j]}$ with a sub-polyline $L_{i,j}$ containing a vertex with degree $> 2$ at $k$, split $CH_{[i,j]}$ at $P_k$, see Fig. 3.3. For a non-planar intersection, split $CH_{[i,j]}$ at endpoints of segment involved in the intersection.

**Heuristic 3.1.** *Given a context hull $CH_{[i,j]}$ of sub-polyline $L_{i,j}$ with vertex $P_k$ where $degree(P_k) > 2$, $k \neq i$, and $k \neq j$, preserve $k$ by splitting $CH_{[i,j]}$ at $k$ into $CH_{[i,k]}$ and $CH_{[k,j]}$.*



(a) context hulls



(b) constrained to planar vertex

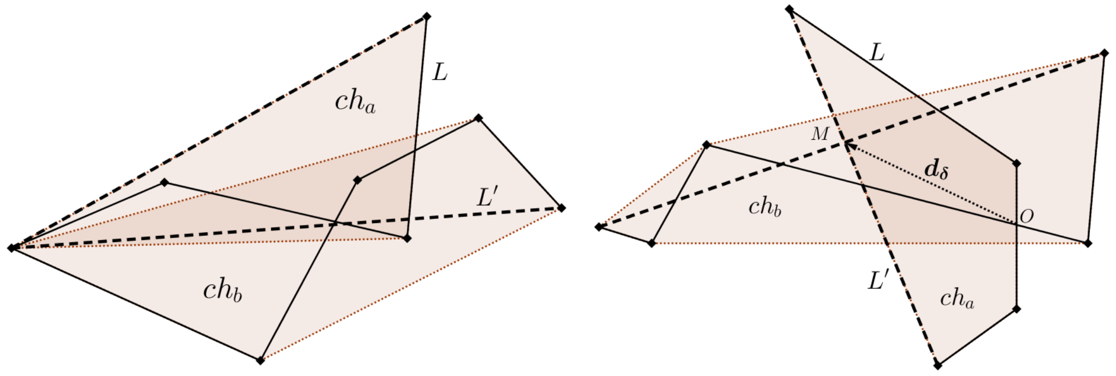
Figure 3.3: Context hulls containing planar intersect (vertex degree $> 2$) (a) and constrained deformation at planar vertex (b)

**Theorem 3.1.** *$\overline{P_iP_j}$ is a consistent collapse of $L_{i,j}$ if the convex vertices of $L_{i,j}$ consists of $P_i$ and $P_j$ and the vertices between $i$ and $j$ are of degree two, where $j - i > 2$.*

*Proof.* The convex vertices of a line segment or a polyline with collinear vertices are the endpoints of the input polyline. The area of $CH_{[i,j]}$ is zero and will not change the shape of $L_{i,j}$ collapsed as $\overline{P_iP_j}$. □

After heuristic 3.1, all vertices with at least degree three are constrained as an endpoint of a context hull. To maintain non-planar intersects, we deform context hulls at endpoints of segments involved in non-planar intersection. We also constrain non-planar relationship between context hulls, see Fig. 3.4a.

**Heuristic 3.2.** *Deform $CH_{[i,j]}$ and $CH_{[m,n]}$ contiguous ($j = m$) at $j$ if $\overline{P_iP_j}$ and $\overline{P_mP_n}$ intersect only at $P_j$ but $L_{i,j}$ and $L_{m,n}$ intersect other than $P_j$, see Fig. 3.4a.*



(a) contiguous (b) non-contiguous

Figure 3.4: Contiguous context hulls with non-planar intersects (a) and overlap of two non-contiguous context hulls (b)

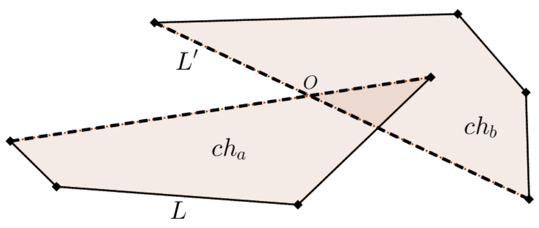
Non-contiguous crossings can result from the same polyline or from other polylines of the same feature class. We define the "relax" distance ($\delta$) as the offset between the intersect of sub-polylines and the intersect of their collapsed generalized segments. $\delta$ is an optional user defined non-planar distance offset at the time of simplification.

**Heuristic 3.3.** *Deform $CH_{[i,j]}$ and $CH_{[m,n]}$, where $j \neq m$ and $n \neq i$, if the distance ($d_\delta$) between $O$ and $M$ is greater than $\delta$, where $O$ and $M$ are the intersects between sub-polylines ($L_{i,j}$,$L_{m,n}$) and collapsed segments ($\overline{P_iP_j}$,$\overline{P_mP_n}$) respectively. See Fig. 3.4b.*

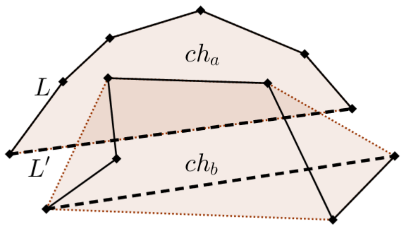
It is possible to refine heuristic 3.3 by choosing one or a combination that yields the smallest $d_\delta$ (Fig. 3.4b): (i) deform $CH_{[i,j]}$ and collapse $CH_{[m,n]}$ as $\overline{P_mP_n}$, (ii) deform $CH_{[m,n]}$ and collapse $CH_{[i,j]}$ as $\overline{P_iP_j}$, and (iii) deform $CH_{[m,n]}$ and deform $CH_{[i,j]}$.

Two non-contiguous hulls can intersect resulting in a simplified crossing but the sub-polylines of their context hulls are disjoint, see an example in Fig 3.5a.

**Heuristic 3.4.** *Given two intersecting non-contiguous context hulls $CH_{[i,j]}$ and $CH_{[m,n]}$ with disjoint sub-polylines $L_{i,j}$ and $L_{m,n}$, deform $CH_{[i,j]}$, if $\overline{P_iP_j}$ intersects $L_{m,n}$ and $\overline{P_mP_n}$.*



(a) inconsistent collapse, deform $ch_b$



(b) consistent collapse

Figure 3.5: Inconsistent collapse of $ch_b$ based on heuristic 3.4 in (a); disjoint sub-polylines with consistent collapsed segments (disjoint) (b)

Two intersecting non-contiguous context hulls $CH_{[i,j]}$ and $CH_{[m,n]}$ can be collapsible. Unlike other algorithms [Saalfeld, 1999; da Silva and Wu, 2006; Corcoran et al., 2011], the presence of a sub-polyline in a context space does not trigger further deformation if its collapse is outside the context space (see Fig. 3.5b).

**Heuristic 3.5.** *Intersecting $CH_{[i,j]}$ and $CH_{[m,n]}$ are collapsible if $L_{i,j}$ and $L_{m,n}$ are disjoint and their resulting collapsed segments ($\overline{P_iP_j}$ and $\overline{P_mP_n}$) are also disjoint.*

It can be deduced from heuristic 3.5 that in the absence of other spatial constraints, it is topologically consistent to collapse disjoint context hulls, see Fig. 3.6.

**Heuristic 3.6.** *Disjoint $CH_{[i,j]}$ and $CH_{[m,n]}$ are topologically collapsible since their collapsed segments $\overline{P_iP_j}$ and $\overline{P_mP_n}$ are also disjoint.*

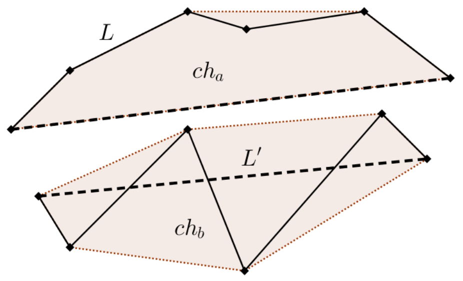


Figure 3.6: Collapsible disjoint context hulls

The sub-polylines of contiguous hulls may not have non-planar intersects as shown in Fig. 3.7, the resulting collapse of each context hull can be an "inverse" or "reflection" at the contiguous vertex. For example, in Fig.3.7, the relative sidedness of $L_{i,k}$ to $L_{k,j}$ is inconsistent with $\overline{P_iP_k}$ to $\overline{P_kP_j}$.

**Heuristic 3.7.** *Deform $CH_{[i,k]}$, if $P_j$ is completely inside $CH_{[i,k]}$ and $CH_{[k,j]}$ if it completely contains $P_i$, where $CH_{[i,k]}$ and $CH_{[k,j]}$ are contiguous at $k$.*

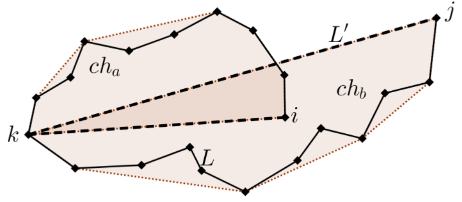


Figure 3.7: Linear inversion problem, $L_{i,k}$ to $L_{k,j}$ is topologically inconsistent with $\overline{P_iP_k}$ to $\overline{P_kP_j}$

Contiguous $CH_{[i,k]}$ and $CH_{[k,j]}$ with intersect at vertex $P_k$ are collapsible relative to each other if their sub-polylines only intersect at $P_k$.

**Heuristic 3.8.** *$CH_{[i,k]}$ and $CH_{[k,j]}$ are collapsible as $\overline{P_iP_k}$ and $\overline{P_kP_j}$ if $P_j$ and $P_i$ are not contained in $CH_{[i,k]}$ and $CH_{[k,j]}$ respectively (see Fig. 3.8)*

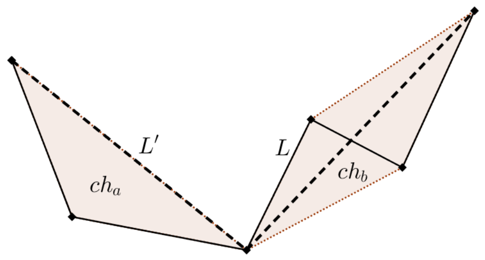


Figure 3.8: Collapsible contiguous context hulls

The disjoint or intersect relation of a sub-polyline to objects in its context space is important. In this paper, we constrain the $\overline{P_iP_j}$ to have the same spatial relation (disjoint/intersect) to context neighbours as $L_{i,j}$ of $CH_{[i,j]}$. For example, in Fig. 3.9, $ch_a$ has two context neighbours $O_A$ and $O_B$, $\overline{P_iP_j}$ should have the same geometric relation as $L_{i,j}$ (disjoint with $O_A$ and intersect $O_B$).

**Heuristic 3.9.** *Deform $CH_{[i,j]}$ if $\overline{P_iP_j}$ has a different disjoint or intersect relation to a context neighbour as expressed by $L_{i,j}$.*

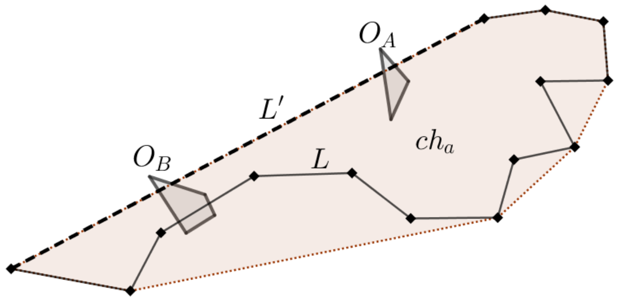


Figure 3.9: Geometric relation (disjoint/intersect). Inconsistent collapse: $\overline{P_iP_j}$ should be disjoint with $O_A$ and intersect $O_B$

The proximity of a simplification to context neighbours can change the semantic representation of a polyline. For example, in marine vessel trajectories, depth and minimum distance to a coastline is important with respect to navigation. In Fig. 3.10, the collapse of $ch_a$ changes the proximity relation of $O_a$ from $\delta_a$ to $\delta_b$ and for $O_b$, $\delta_c$ changes to $\delta_d$.

**Heuristic 3.10.** *Deform $CH_{[i,j]}$ if $\delta_s < \delta$ and $\delta_o \geq \delta$, where $\delta_s$ is the minimum distance between $\overline{P_iP_j}$ and a context object ($O$), $\delta_o$ is the minimum distance between $L_{i,j}$ and $O$. $CH_{[i,j]}$ is collapsible as $\overline{P_iP_j}$ if $\delta_o < \delta$ since $\delta_s$ can be $< \delta_o$ or $\geq \delta_o$.*

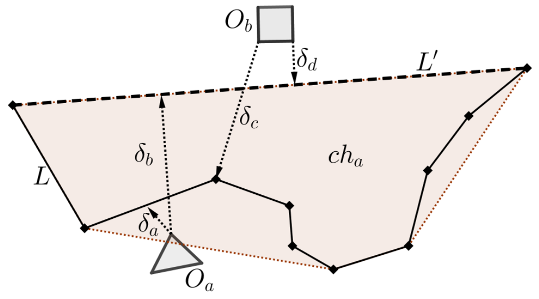
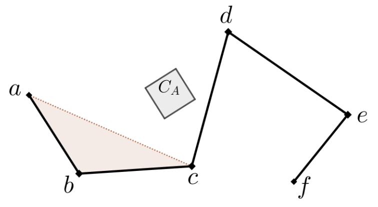


Figure 3.10: Minimum distance relation to context neighbours

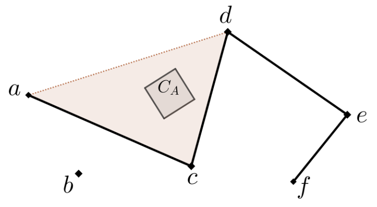
Keeping the endpoints fixed, $L_{i,j}$ is homotopic to $\overline{P_iP_j}$ if $L_{i,j}$ can be continuously deformed into $\overline{P_iP_j}$. The deformation of vertices between $i$ and $j$ are removed without leaving the planar space. Context neighbours are obstacles to such a continuous deformation. By constraining each $L_{i,j}$ to be homotopic (strongly homotopic [Abam et al., 2014]) to $\overline{P_iP_j}$, a chain of homotopic collapsed segments ($L'$) is homotopic to the original polyline ($L$). Our implementation unlike [Abam et al., 2014; Cabello et al., 2004] is not limited by collinear vertices (horizontal or vertical) and self-intersections in a polyline. Our homotopy algorithm also supports arbitrary geometries as context objects.

Homotopy in the context of points, lines, and polygons is achieved by relaxing the constraint on the boundary of such context neighbours: if $L_{i,j}$ intersects $O_a$, $\overline{P_iP_j}$ is considered homotopic if it intersects $O_a$. $L_{i,j}$ and $\overline{P_iP_j}$ are not homotopic if they have a different intersect relation to $O_a$. We develop a chain deformation algorithm to test the homotopic deformation of $L_{i,j}$ to $\overline{P_iP_j}$ using disjoint context neighbours. Our chain data structure is like the classic doubly link list with each vertex having a pointer to the next and previous vertex. All context neighbours are indexed in an in-memory R-Tree to speed up processing.

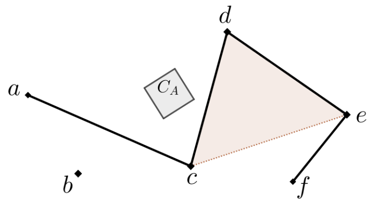
Keeping $i$ and $j$ fixed, we iterate over the chain starting at $i + 1$. Let $i = a$, $i + 1 = b$, $i + 2 = c$ and so on, starting at $b$, form a triangle with the previous and next vertices of $b$ as $T_{abc}$ (see Fig. 3.11a). At $b$, find context neighbours that intersect $T_{abc}$ from the R-Tree. If $T_{abc}$ is disjoint from context objects, the vertex at $b$ is removed to collapse $T_{abc}$ as $\overline{P_aP_c}$, else, the iterator advances to $c$ with previous and next vertices as triangular vertices. In Figure 3.11b, vertex $c$ is not removable because of $C_A$, the iterator advances to $d$, consecutive triangles $T_{cde}$ and $T_{cef}$ are collapsible. We repeat the iteration if a triangle was collapsed in the previous iteration and the number of vertices in the chain is greater than two. In Fig. 3.11e, the second iteration of triangular deformation will collapse $T_{acf}$ by removing $c$. The algorithm eventually terminates since $a$ and $f$ have no previous and next vertices respectively. If there are intermediate vertices between the endpoints and an iteration does not yield a triangular collapse, it means all the edges are constrained by context neighbours in such a way that $\overline{P_iP_j}$ is not homotopic to $L_{i,j}$.



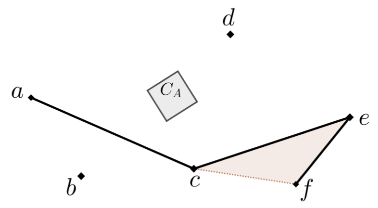
(a) collapse triangle $abc$, remove vertex $b$, advance to $c$



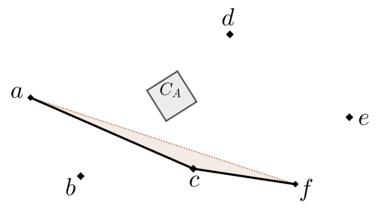
(b) triangle $acd$ is not collapsible, advance to $d$



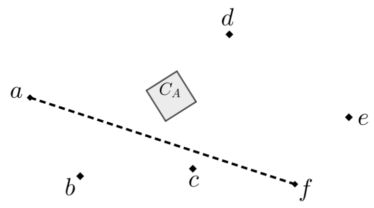
(c) collapse triangle $cde$, remove vertex $d$, advance to $e$



(d) collapse triangles $cef$, advance to $f$, stop iteration 1



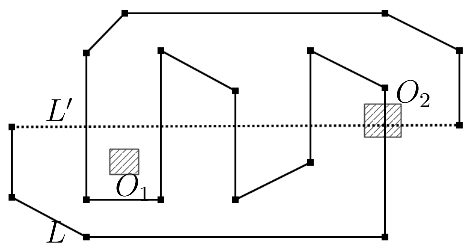
(e) collapse triangle $acf$, remove vertex $c$, advance to $f$



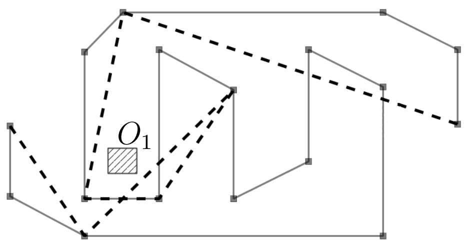
(f) stop iteration 2 - endpoints $a$ and $f$

Figure 3.11: Homotopic chain deformation, second iteration starts at (e)

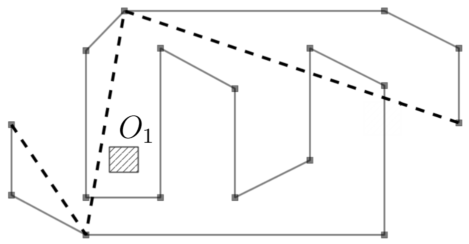
In Fig. 3.12, we show the gradual shape deformation at each iteration with a dash polyline. Let $L_{i,j}$ have $n$ vertices, keeping $i$ and $j$ fixed, we have $n-2$ vertices to collapse in $k$ iterations. The worst case of the chain deformation algorithm is to collapse only one triangle per iteration, thus $k = n - 2$. The complexity of computing the homotopy of each context hull is $\mathcal{O}(n^2)$, where $n$ is the number of vertices in $L_{i,j}$.



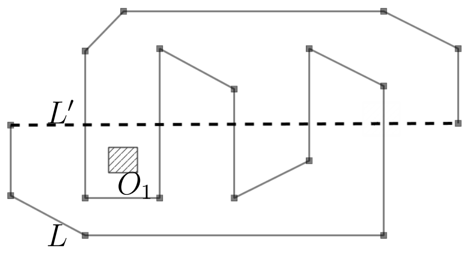
(a) polyline, filter for disjoint neighbours



(b) deformation (dash line) - iteration 1



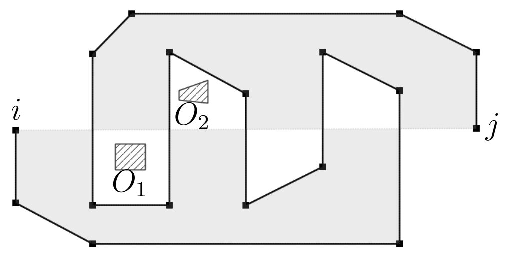
(c) deformation (dash line) - iteration 2



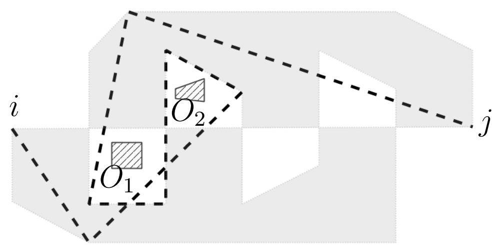
(d) deformation (dash line) - iteration 3

Figure 3.12: Homotopic chain deformation

Corcoran et al. [2011] and Funke et al. [2017] based on the work of de Berg et al. [1998] consider $L_{i,j}$ not homotopic to $\overline{P_i P_j}$ if a context object is in the bounded face of the polygon formed by $L_{i,j}$ and $\overline{P_i P_j}$. The polygon edge $\overline{P_j P_i}$ closing the loop from $j$ to $i$ can create a complex polygon by intersecting some of the edges of $L_{i,j}$. Algorithms [de Berg et al., 1998; Corcoran et al., 2011; da Silva and Wu, 2006; Funke et al., 2017] in literature use point-in-polygon (even-odd rule by Shimrat [1962]) to test if a context object (point) is inside or outside a bounded face of a polygon. This approach does not extend well to arbitrary (possibly self intersecting) planar objects such as polylines and polygons. Moreover, two points in non-bounded faces of a complex polygon can still prevent a homotopic deformation of $L_{i,j}$ into $\overline{P_i P_j}$. It is therefore not sufficient to test the inside versus outside relation of each context neighbour, see Figure 3.13.



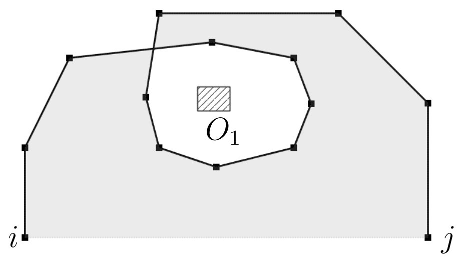
(a) simple polyline ($L_{i,j}$) as a complex polygon ($Q_{ij}$) formed by vertices of $L_{i,j}$ and $\overline{P_j P_i}$. The two context objects ($O_1$ and $O_2$) are not bounded by $Q_{ij}$



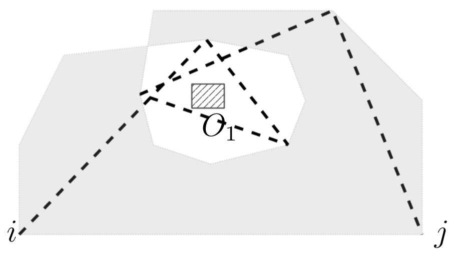
(b) planar deformation of $L_{i,j}$ is restricted by non-bounded context objects ($O_1$ and $O_2$)

Figure 3.13: Restricted non-bounded context objects in chain deformation

An arbitrary ring of vertices $L_{i,j}$ ($P_i, P_{i+1}, ..., P_j, P_i$) with non-planar self-intersection can create non-bounded faces in a polygon (Fig. 3.14). Our method prevents passing over context objects in this non-bounded region. See Figure 3.14.



(a) non-planar self-intersecting polyline



(b) restricted by non-bounded context object ($O_1$)

Figure 3.14: Restricted self-intersecting chain deformation by a non-bounded context object

**Heuristic 3.11.** *Deform $CH_{[i,j]}$ if the chain $L_{i,j}$ cannot be continuously deformed as $\overline{P_i P_j}$ with respect to disjoint context geometries.*

### 3.5.2 Constrained Implementation

Using the RDP algorithm described in section 3.4, each polyline is decomposed into a set of contiguous context hulls $S_h$ ($CH_{[i,g]}$, $CH_{[g,k]}$, ..., $CH_{[m,j]}$) at a pre-specified $\varepsilon$. Based on a set of constrained simplification options, context hulls in $S_h$ are split at planar and non-planar self-intersections. $S_h$ is then bulk-loaded into an in-memory R-Tree ($H_{db}$). To speed up finding $k$-nearest context neighbours, other planar geometries are bulk-loaded into another in-memory R-Tree ($C_{db}$). User defined constrained simplification options include the following:

1. simplification threshold ($\varepsilon$) (number, e.g., RDP),
2. relaxation distance (number, $\delta$ in heuristic 3.3),
3. avoid new self-intersects (boolean, heuristic 3.5),
4. geometric relation (boolean, heuristic 3.9),
5. minimum distance (number, $\delta$ in heuristic 3.10),
6. distance relation (boolean, heuristic 3.10),
7. homotopy relation (boolean, heuristic 3.11), and
8. keep planar self-intersects (boolean, heuristic 3.1).
9. keep non-planar self-intersects (boolean, heuristics 3.2 and 3.3).

Based on the options provided, a constrained simplification algorithm is described in Algorithm 3.1.

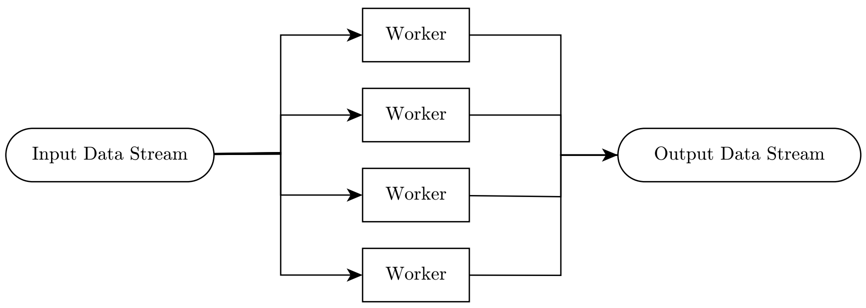


Figure 3.15: Concurrent processing model in Algorithm 3.1. A worker refers to a concurrent *go* routine

**Algorithm 3.1** Simplify at $\varepsilon$

```
1. Let $S_h$ be all context hulls in $H_{db}$.
2. Using each context hull ($CH_{[i,j]}$) in $S_h$, query $H_{db}$ for hull neighbours and $C_{db}$ for
   planar context objects, find all deformable hulls based on simplification options;
   Check for collapsibility based on the heuristics outlined in section 3.5.1. Results of
   context hulls that require further deformation are stored in set $S_d$.
3. Deform each context hull in $S_d$ based on RDP (see section 3.4) and store results in
   set $S_s$.
4. Remove from $H_{db}$ all hulls in $S_d$ (parent hulls of $S_s$).
5. Update $H_{db}$ by bulk-loading context hulls in $S_s$.
6. Let $S_h$ be $S_s$, if $S_h$ is not empty, repeat step 2, else, terminate and merge segments
   that are mergeable in $H_{db}$.
```

Algorithm 3.1 is implemented using the *Go* programming language. Step 2 and 3 in Algorithm 3.1 uses eight (8) concurrent routines based on the layout in Fig. 3.15 to speed up processing.

## 3.6 Experimental Evaluation

To demonstrate the effectiveness of our constrained simplification algorithm, we have implemented and tested its performance with varied datasets. In this paper, we use a road network and elevation contours. Experiments are performed on an Intel® Core™ i7 3.6GHz x4, 16 GB RAM. To accelerate processing, all computations are done in memory ($10GB$ available RAM).

The first dataset used for experimental evaluation is the New Brunswick (NB) Road Network, Canada. A Road network has varied topological relations that require preservation: self-intersection at junctions, non-planar intersection at overpasses, disjoint relation in two parallel roads, and many more. The dataset consists of $67,484$ polylines with $1,293,345$ vertices (see appendix. A.1). The second dataset consists of USGS elevation contours (40$m$ interval) for Pitkin County, Colorado, USA. The contour dataset contains $42,701$ polylines with $11,046,779$ vertices. Contours unlike roads present a limited set of requirements: a single contour (polyline or ring) does not self-cross or cross other contour lines (see appendix. B.1).

Simplifications are performed progressively over a range of $\varepsilon$ values. The New Brunswick road network is simplified from $2m$ to $20m$ at an interval of $2m$. The contour data from Pitkin County are simplified starting $5m$ to $40m$ at $5m$ interval (considering only planar $x, y$ without $z$ dimension).

### 3.6.1 Results

At a given $\varepsilon > 0$, each polyline is decomposed into a set of contiguous $\varepsilon$-feasible sub-polylines based on RDP. For each polyline and between polylines, we compute planar self-intersections (vertices with degree greater than two); let these planar vertices of degree greater than two be the set $CV$. Vertices of segments that form non-planar intersections are also added to $CV$. Each $\varepsilon$-feasible sub-polyline $L_{i,j}$ is split into sub-polylines $L_{i,k}$ and $L_{k,j}$ if $i < k < j$, where $k$ is the index of a vertex contained in $CV$ (see heuristic 3.1). The proximity property of RDP ensures that subsequent deformation of $\varepsilon$-sub-polylines are within a distance $\varepsilon$ of the original polyline.

Isolated RDP decomposition of each polyline can result in topological and spatial errors. To resolve these errors in the context of other planar objects, all the sub-polylines are contextually deformed based on algorithm 3.1. At the start, each sub-polyline is considered "deformable" (invalid). Algorithm 3.1 is used to filter which sub-polylines are collapsible in the context of planar constraints. Each iteration of algorithm 3.1 reduces the number of deformable sub-polylines. The progression of number of deformable units versus iterations is shown in Figures 3.16 and 3.17. The number of iterations required to resolve all conflicting sub-polylines increases with increasing $\varepsilon$, because the area of a context hull depends on the size of $\varepsilon$. A bigger context area increases the probability of topological or homotopy invalidation during simplification. Figures 3.16 and 3.17 show a fast convergence from the initial number of deformable hulls at given $\varepsilon$.

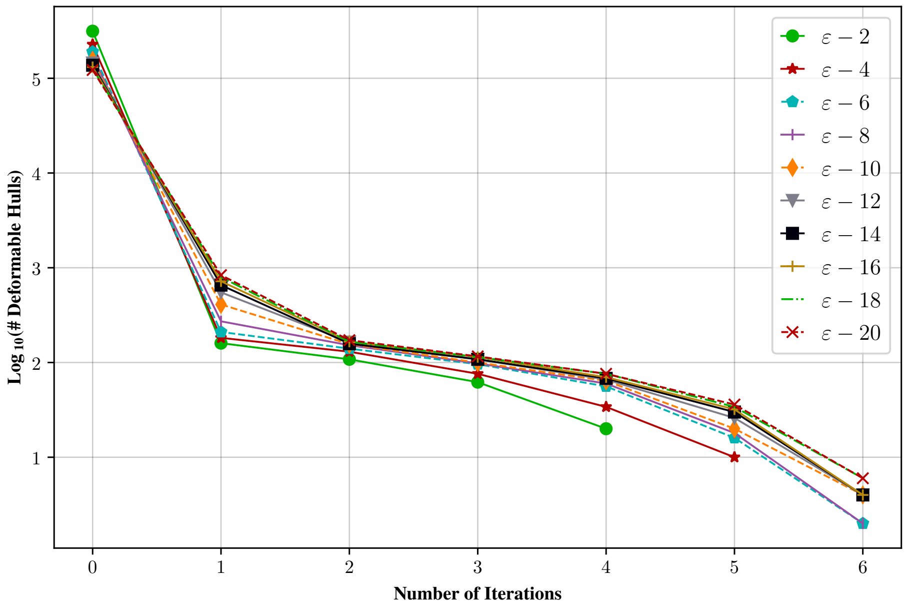


Figure 3.16: Deformation convergence - Road Network - New Brunswick

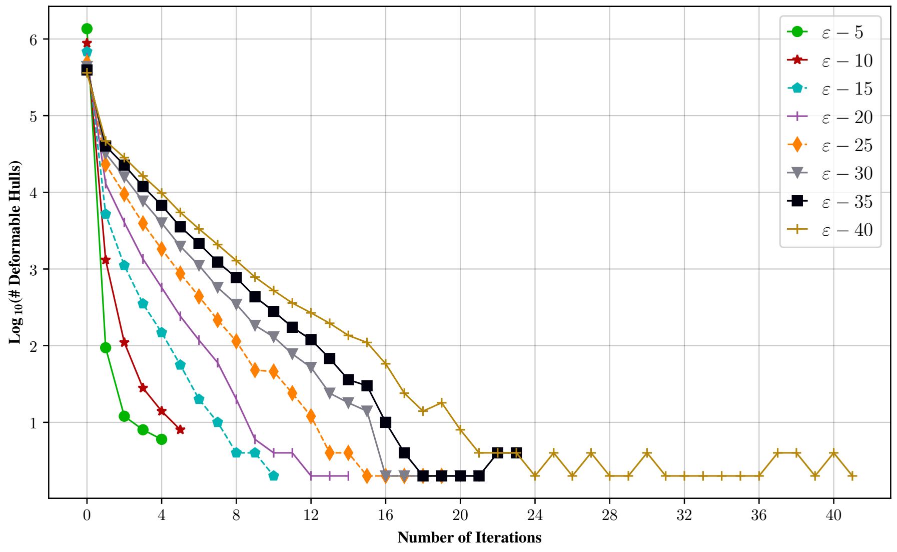


Figure 3.17: Deformation convergence - Contours - Pitkin County

Tables 3.1 and 3.2 show the running time and percentage compression of constrained and unconstrained simplification of NB road network and Pitkin contours respectively. It can be observed unconstrained RDP is faster than constrained RDP. The running time of constrained RDP consists of the running time of unconstrained plus additional processing to resolve simplification errors. The percentage of data reduction is computed as $(1 - \frac{N_s}{N_o}) * 100\%$ where $N_s$ and $N_o$ are the number of vertices in the simplified and original dataset respectively. Constrained RDP in Tables 3.1 and 3.2 show competitive (small difference) compression ratios compared to unconstrained RDP simplification. The difference in percentage compression for the road network between constrained and unconstrained simplification is quite small (see Tables 3.1), this is because most road outlines are straight lines with less abrupt changes in direction. A bigger difference will be observed in detailed outlines (e.g., meandering river or coastlines) with topological constraints.

Figures 3.18 and 3.19 graphically highlight limitations of unconstrained compared to constrained simplification. Figure 3.18b introduces a self-intersection whereas Figure 3.18c preserves the original disjoint relation in Figures 3.18a. In the context topologically disjoint contour lines (Fig. 3.19a), isolated unconstrained simplification of each contour at $\varepsilon$ can introduce self-intersections between neighbouring contours, see Figure 3.19b. Constrained simplification of the same scene in Fig. 3.19a is shown in Fig. 3.19c. Fig. 3.19c preserves the topology relation in Figure 3.19a.

Table 3.1: Road Network - Province of New Brunswick ($67,484$ polylines - $1,293,345$ vertices)

| $\varepsilon$ (meters) | Constrained % Compression | Unconstrained % Compression | Constrained Time(seconds) | Unconstrained Time(seconds) |
|---|---|---|---|---|
| 2 | 70.5 | 70.6 | $35.4 \pm 3.7$ | $5.3 \pm 0.1$ |
| 4 | 77.2 | 77.3 | $25.4 \pm 0.1$ | $3.5 \pm 0.1$ |
| 6 | 80.0 | 80.2 | $22.0 \pm 0.1$ | $2.9 \pm 0.1$ |
| 8 | 81.7 | 81.8 | $20.1 \pm 0.1$ | $2.6 \pm 0.1$ |
| 10 | 82.8 | 82.9 | $19.0 \pm 0.1$ | $2.4 \pm 0.1$ |
| 12 | 83.5 | 83.7 | $17.9 \pm 0.1$ | $2.3 \pm 0.0$ |
| 14 | 84.1 | 84.3 | $17.2 \pm 0.1$ | $2.3 \pm 0.1$ |
| 16 | 84.6 | 84.8 | $16.8 \pm 0.1$ | $2.2 \pm 0.1$ |
| 18 | 85.0 | 85.1 | $16.2 \pm 0.1$ | $2.1 \pm 0.1$ |
| 20 | 85.3 | 85.4 | $15.8 \pm 0.0$ | $2.0 \pm 0.0$ |

Table 3.2: USGS Elevation Contours - Pitkin County ($42,701$ polylines - $11,046,779$ vertices)

| $\varepsilon$ (meters) | Constrained % Compression | Unconstrained % Compression | Constrained Time(seconds) | Unconstrained Time(seconds) |
|---|---|---|---|---|
| 5 | 87.3 | 87.3 | $282.2 \pm 1.4$ | $23.7 \pm 0.4$ |
| 10 | 91.6 | 91.6 | $207.1 \pm 0.9$ | $15.4 \pm 0.5$ |
| 15 | 93.4 | 93.4 | $183.3 \pm 0.3$ | $11.9 \pm 0.1$ |
| 20 | 94.4 | 94.5 | $170.0 \pm 0.4$ | $9.7 \pm 0.3$ |
| 25 | 95.0 | 95.2 | $177.8 \pm 0.4$ | $9.1 \pm 0.3$ |
| 30 | 95.4 | 95.7 | $182.1 \pm 0.6$ | $8.0 \pm 0.0$ |
| 35 | 95.6 | 96.0 | $191.6 \pm 0.3$ | $7.4 \pm 0.2$ |
| 40 | 95.8 | 96.3 | $198.3 \pm 0.4$ | $6.7 \pm 0.1$ |

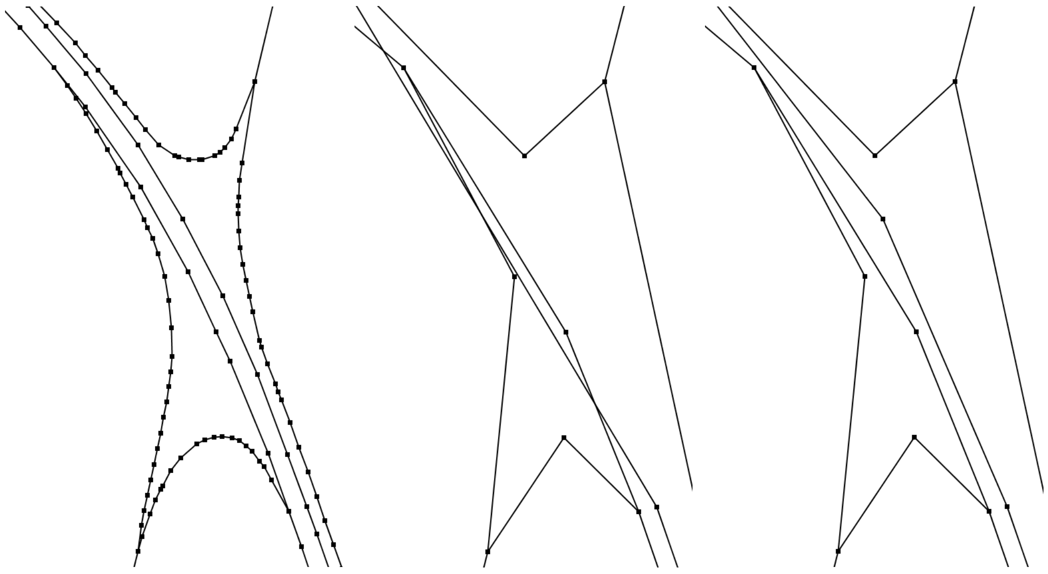

(a) Original (b) Unconstrained (c) Constrained

Figure 3.18: Original Road Network (a), Unconstrained (b) and Constrained (c) RDP Simplification at $\varepsilon = 20m$.

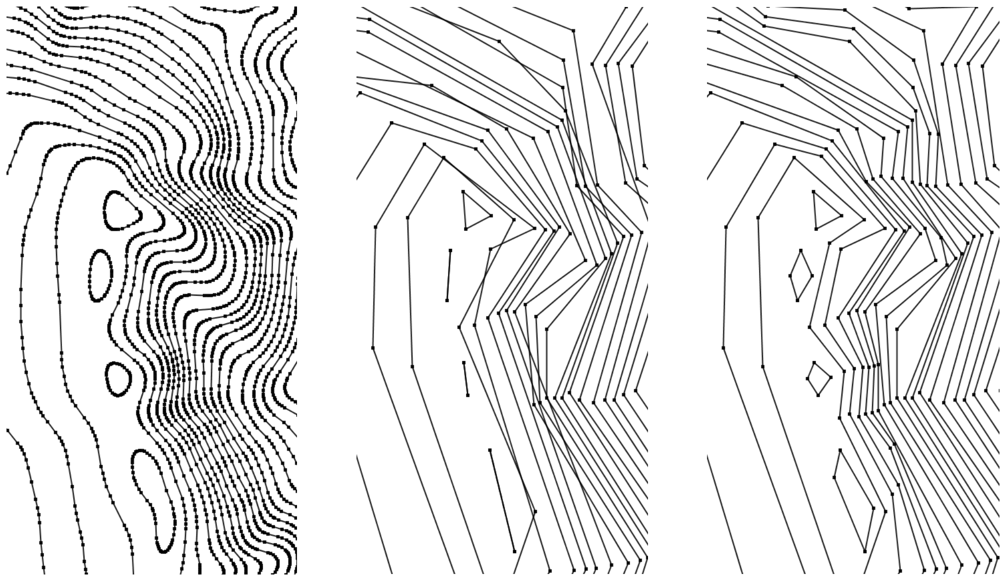

(a) Original (b) Unconstrained (c) Constrained

Figure 3.19: Original Contour (a), Unconstrained (b) and Constrained (c) and RDP Simplification at $\varepsilon = 30m$.

## 3.7 Conclusion

In this paper, we consider the $Min{-}\#$ simplification problem using Ramer [1972] and Douglas and Peucker [1973] (RDP) algorithm. Through a novel set of geometric heuristics, we demonstrate how to avoid self-intersection introduced by the RDP algorithm, preserve planar and non-planar intersection, and constrain spatial relations (geometric, distance, homotopy) to other planar objects (points, polylines, polygons). The algorithms are evaluated on a road network and contour lines to demonstrate its performance and correctness. The results show a fast convergence to a consistent simplification with competitive compression ratio as compared to the unconstrained algorithm. All processing and data structures are implemented in-memory; this is not scalable for datasets that will not fit in main memory (random-access memory - RAM). Future work should explore an external I/O efficient implementation for constrained polyline simplification. The main contribution (section 3.3.1) in this paper is resolving multiple requirements (section 3.5) towards automated line simplification to support cartographic generalization, graphic visualization, and data reduction. Source code of our implementation is available at *github.com/TopoSimplify*.

# References

Abam, M. A., Daneshpajouh, S., Deleuran, L., Ehsani, S., and Ghodsi, M. (2014). Computing homotopic line simplification. *Computational Geometry*, 47(7):728–739.

Agarwal, P. K. and Varadarajan, K. R. (2000). Efficient algorithms for approximating polygonal chains. *Discrete & Computational Geometry*, 23(2):273–291.

Buttenfield, B. (1985). Treatment of the cartographic line. *Cartographica: The Interna-*

*tional Journal for Geographic Information and Geovisualization*, 22(2):1–26.

Cabello, S., Liu, Y., Mantler, A., and Snoeyink, J. (2004). Testing homotopy for paths in the plane. *Discrete & Computational Geometry*, 31(1):61–81.

Chan, W. and Chin, F. (1992). Approximation of polygonal curves with minimum number of line segments. In *International Symposium on Algorithms and Computation*, pages 378–387. Springer.

Corcoran, P., Mooney, P., and Bertolotto, M. (2012). Line simplification in the presence of non-planar topological relationships. In *Bridging the Geographic Information Sciences*, pages 25–42. Springer.

Corcoran, P., Mooney, P., and Winstanley, A. (2011). Planar and non-planar topologically consistent vector map simplification. *International Journal of Geographical Information Science*, 25(10):1659–1680.

Cromley, R. G. (1991). Hierarchical methods of line simplification. *Cartography and Geographic Information Systems*, 18(2):125–131.

da Silva, A. C. G. and Wu, S.-T. (2006). A robust strategy for handling linear features in topologically consistent polyline simplification. In *VIII Brazilian Symposium on Geoinformatics, 19-22 November, Campos do Jordão, São Paulo, Brazil*, pages 19–34.

de Berg, M., van Kreveld, M., and Schirra, S. (1998). Topologically correct subdivision simplification using the bandwidth criterion. *Cartography and Geographic Information Systems*, 25(4):243–257.

Douglas, D. H. and Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122.

Estkowski, R. (1998). No steiner point subdivision simplification is np-complete. *In Proceedings of the 10th Canadian Conference on Computational Geometry*.

Estkowski, R. and Mitchell, J. S. (2001). Simplifying a polygonal subdivision while keeping it simple. In *Proceedings of the seventeenth annual symposium on Computational geometry*, pages 40–49. ACM.

Funke, S., Mendel, T., Miller, A., Storandt, S., and Wiebe, M. (2017). Map simplification with topology constraints: Exactly and in practice. In *2017 Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 185–196. SIAM.

Goodrich, M. T. (1995). Efficient piecewise-linear function approximation using the uniform metric. *Discrete & Computational Geometry*, 14(4):445–462.

Guibas, L. J., Hershberger, J. E., Mitchell, J. S., and Snoeyink, J. S. (1993). Approximating polygons and subdivisions with minimum-link paths. *International Journal of Computational Geometry & Applications*, 3(04):383–415.

Heckbert, P. S. and Garland, M. (1997). Survey of polygonal surface simplification algorithms. Technical report, Carnegie-Mellon Univ. Pittsburgh PA., School of Computer Science.

Hershberger, J. and Snoeyink, J. (1992). Speeding up the Douglas-Peucker line-simplification algorithm. Technical report, Vancouver, BC, Canada, Canada.

Hobby, J. D. (1993). Polygonal approximations that minimize the number of inpections. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 93–102.

Imai, H. and Iri, M. (1988). Polygonal approximations of a curve — formulations and algorithms. In Toussaint, G. T., editor, *Computational Morphology*, volume 6 of *Machine Intelligence and Pattern Recognition*, pages 71–86. North-Holland.

Kulik, L., Duckham, M., and Egenhofer, M. (2005). Ontology-driven map generalization. *Journal of Visual Languages & Computing*, 16(3):245–267.

Li, Z. and Openshaw, S. (1992). Algorithms for automated line generalization1 based on a natural principle of objective generalization. *International Journal of Geographical Information Systems*, 6(5):373–389.

Li, Z. and Openshaw, S. (1993). A natural principle for the objective generalization of digital maps. *Cartography and Geographic Information Systems*, 20(1):19–29.

Mantler, A. and Snoeyink, J. (2000). Safe sets for line simplification. In *10th Annual Fall Workshop on Computational Geometry*.

McMaster, R. B. (1987a). Automated line generalization. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 24(2):74–111.

McMaster, R. B. (1987b). The geometric properties of numerical generalization. *Geographical Analysis*, 19(4):330–346.

Melkman, A. and O'Rourke, J. (1988). On polygonal chain approximation. In Toussaint, G. T., editor, *Computational Morphology*, volume 6 of *Machine Intelligence and Pattern Recognition*, pages 87–95. North-Holland.

Miyaoku, K. and Harada, K. (1998). Approximating polygonal curves in two and three dimensions. *Graphical Models and Image Processing*, 60(3):222–225.

Muller, J. (1990). The removal of spatial conflicts in line generalization. *Cartography and Geographic Information Systems*, 17(2):141–149.

Ramer, U. (1972). An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing*, 1(3):244–256.

Saalfeld, A. (1999). Topologically consistent line simplification with the Douglas-Peucker algorithm. *Cartography and Geographic Information Science*, 26(1):7–18.

Saalfeld, A. (2000). Complexity and intractability: limitations to implementation in analytical cartography. *Cartography and Geographic Information Science*, 27(3):239–250.

Shi, W. and Cheung, C. (2006). Performance evaluation of line simplification algorithms for vector generalization. *The Cartographic Journal*, 43(1):27–44.

Shimrat, M. (1962). Algorithm 112: position of point relative to polygon. *Communications of the ACM*, 5(8):434.

Suri, S. (1986). A linear time algorithm for minimum link paths inside a simple polygon. *Computer Vision, Graphics, and Image Processing*, 35(1):99–110.

Tienaah, T., Stefanakis, E., and Coleman, D. (2015). Contextual Douglas-Peucker simplification. *Geomatica*, 69(3):327–338.

Weibel, R. (1997). Generalization of spatial data: Principles and selected algorithms. *Algorithmic foundations of geographic information systems*, pages 99–152.

White, E. R. (1985). Assessment of line-generalization algorithms using characteristic points. *The American Cartographer*, 12(1):17–28.

Wu, S.-T. and Marquez, M. R. G. (2003). A non-self-intersection Douglas-Peucker algorithm. In *Computer Graphics and Image Processing, 2003. SIBGRAPI 2003. XVI Brazilian Symposium on*, pages 60–66. IEEE.

# Chapter 4:

# Topologically Consistent Online Trajectory Simplification

## 4.1 Abstract

We study online simplification of arbitrary massive trajectories in the context of other planar objects. A trajectory consists of possibly an infinite number of incoming spatiotemporal instants: $(x_0, y_0, t_0)$, $(x_1, y_1, t_1)$, $(x_i, y_i, t_i)$, ... of a moving object. Our online implementation is constrained to avoid and preserve self-intersection between simplification units. Other contextual constraints include disjoint/intersect, homotopy, and proximity in the context of planar objects. Using real world data, we evaluate our implementation using moving vessels in the Aegean Sea with islands as planar constraints. Unconstrained simplification in our evaluation show a higher compression ratio compared to constrained simplification. Our results show a compression difference of about $4\%$ between constrained and unconstrained simplification. Constrained simplification also took less than four times the processing time of unconstrained simplification. The results show a competitive compression ratio in a reasonable amount of time to resolve spatial and topological errors in

the constrained version.

Unconstrained trajectory simplification is fast with a higher compression ratio but leads to spatial and topological errors. A consistent constrained simplification provides the benefits of data reduction while preserving the original spatiotemporal characteristics. Our implementation provides the first treatment of an online trajectory simplification that avoids self-intersection as a result of polyline simplification, preserves non-planar self-intersection between simplification units and maintains homotopy in the context of arbitrary planar objects. We also provide a framework for concurrent streaming and constrained simplification to answer ad hoc queries during movement.

## 4.2 Introduction

Recent advancements in location based sensors (Global Positioning System (GPS), Radio-Frequency Identification (RFID)), wearable devices (smart watches, GPS enabled sports bands), hand held devices (cell phones), and self navigation robots (self drivings cars, drones) have lead to an increasing timestamped telemetry of moving objects.

A trajectory $M$ of a moving object can be tracks of a human, animal, a vehicle with GPS navigation (car or vessel), hurricane, or the spatiotemporal trace of any moving entity. The trace of $M$ can be sampled based on time (e.g., at every 10 minutes), distance (e.g., at every 10 meters), deflections (e.g., deflection angle greater than 10 degrees) and so on. Because of storage and transmission limitations, the spatiotemporal locations of a moving object is generalized by discretization at data collection. It is beneficial to collect data at the highest resolution and then reduce it in order to optimize performance in subsequent data processing, transmission, visualization and/or management applications.

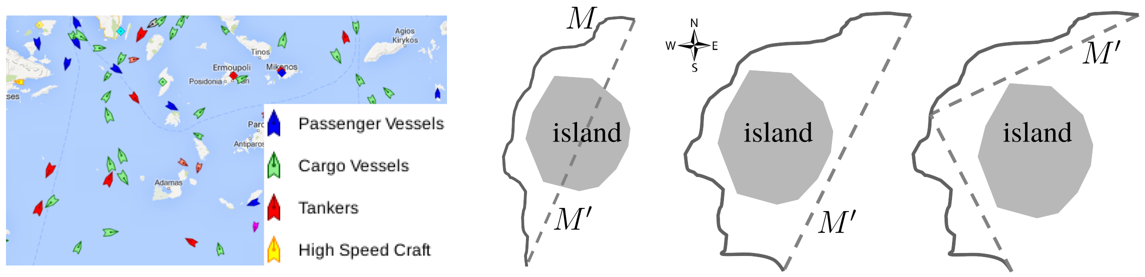
Piecewise linear approximation is one of the most widely used methods in data reduction due to its algorithmic simplicity and low computational complexity [Meratnia and Rolf, 2004]. Piecewise linear approximation algorithms can be classified as *batch* or *online*.

Batch algorithms require all spatiotemporal instants of $M$, whereas online do not, and are used to simplify trajectories in a real-time environments.

The spatiotemporal instants of $M$ are represented as $(l_0, t_0)$, $(l_1, t_1)$, $(l_i, t_i)$, ..., $(l_n, t_n)$), where $l_0$, $l_1$, $l_i$,..., $l_n$ are sampled locations and $t_0$, $t_1$, $t_i$, ..., $t_n$ are corresponding time stamps. A piecewise linear approximation of $M$ assumes that successive instants of $M$ are connected by straight line segments. Based on an approximation criteria, a *restricted* simplification of $M$ seeks to find a sub-series $M'$ of $M$ by keeping some instants of $M$ without displacement or introduction of new spatiotemporal instants that are not part of $M$. The vertices of $M'$ is therefore a subset of $M$. The simplification error $\varepsilon$ is the maximum separation between $M$ and $M'$. The utility of $M'$ is lost if it does not capture the relevant spatiotemporal characteristics (shape, speed, direction, and other attributes) of $M$ [Meratnia and Rolf, 2004; Potamias et al., 2006]. In this paper, we focus on constrained online trajectory simplification in the context of arbitrary objects (points, lines, or polygons) as planar constraints.

Trajectory simplification algorithms that extend piecewise linear approximation to handle the temporal dimension inherit the drawbacks of such techniques. Line simplification is a well studied problem in cartography, computer graphics, and computational geometry. The following authors have identified spatial and topology errors: Muller [1990], Saalfeld [1999], Estkowski and Mitchell [2001], Corcoran et al. [2011], Daneshpajouh and Ghodsi [2011], Stefanakis [2012], and Tienaah et al. [2015]. Moving objects in the physical world have spatial and temporal relations to other objects or their path of movement (cars are constrained by the road network and other moving vehicles; vessels at sea are constrained by landmasses and navigable routes).

Trajectory simplification in isolation (disregarding spatial relations to planar objects or trajectory fragments) can lead to topology and proximity errors. Using trajectories of vessels at sea (Marine Traffic: *marinetraffic.com*), we illustrate three simplification problems that can arise if trajectories are simplified independent of spatial relations to islands. Sea vessel trajectories in the context of islands should not intersect land and a simplification should have the same topological relation to islands as the original. The simplified polyline (dashed) in Fig. 4.1b violates the disjoint relation of the original trajectory. In Fig. 4.1c, the simplification is on a different side of the island compared to the original trajectory (different homotopy class), and Fig. 4.1d shows the simplification may violate keeping a certain minimum distance to the island.



(a) Vessels and Islands (b) Disjoint error (c) Homotopy error (d) Proximity error

Figure 4.1: (a) Sea vessels moving between islands (a snapshot from Marine Traffic in the Aegean). The source (original in solid polyline) and target (simplified in dashed line) trajectories of a vessel in spatial relation to an island (polygon). Violation of spatial relations: (b) topological as the target intersects the island; (c) homotopy as the target passes through the eastern side of the island; and (d) distance as the target is too close to the coastline.

The focus of this paper is the problem of finding a restricted sub-series of a trajectory without moving timestamped positions of the original or introducing new instants. We further limit our scope to online-simplification with spatial relation constraints to objects that share the same planar space as the trajectory. We focus on online simplification by creating an environment that can handle concurrent streams of timestamped positions from moving vessels at sea. Our implementation is to support a real-time monitoring system that is able to provide real-time responses to the simplification state of a moving object. The contributions of this paper include:

1. avoids self-intersection as a result of simplification,

2. preserves non-planar self-intersection between simplification units,

3. preserves intersect, disjoint, distance, and homotopy relation to arbitrary context geometries, and

4. provides external simplification of arbitrary trajectories in an online environment.

The remainder of this paper is organized into four major sections. In section 4.3, we review related work on sampling trajectory streams. In section 4.4 we review and develop our simplification algorithm. Experimental evaluation of algorithms are discussed in section 4.6. The paper concludes with contribution and future work in section 4.7.

## 4.3 Related Work

For a detailed survey of line simplification algorithms and their development over the years, refer to [Heckbert and Garland, 1997; Shi and Cheung, 2006]. One of the popular algorithms that extended to spatiotemporal simplification is the Rammer-Douglas-Pecker (RDP) algorithm [Ramer, 1972; Douglas and Peucker, 1973]. McMaster [1987] and White [1985] showed that the RDP algorithm outperforms (mathematical and psychological) other simplification algorithms. The RDP algorithm recursively partitions a polyline based on a given simplification error $\varepsilon$ of vertices from a generalized segment (line joining the first and last vertex). Vertices with Euclidean distance less than $\varepsilon$ are discarded and characteristic or critical vertices of a polyline are kept. Meratnia and Rolf [2004] extended the RDP algorithm to consider the spatiotemporal characteristics of a trajectory by replacing the simplification error ($\varepsilon$) with a time ratio function - Synchronized Euclidean Distance (SED) or Time Ratio (TR).

In the spatiotemporal domain, the SED distance function projects a temporal position onto a generalized line segment, the metric that measures the instant location is the Euclidean distance between the projected and its original location. This approach has been explored by Cao et al. [2006] and Muckell et al. [2011]. A detailed description of the SED algorithm is discussed in section 4.4.1.1. Other metrics for trajectories simplification include: position, speed, velocity, direction [Long et al., 2013; Muckell et al., 2011, 2014; Lin et al., 2016; Potamias et al., 2006]. Other techniques such similarity based compression have been explored by Birnbaum et al. [2013].

Lossy and lossless trajectory compression for a good compression ratio and small error margin has developed by Nibali and He [2015]. A bottom-up multi-resolution trajectory simplification has been explored by Chen et al. [2012]. For a comprehensive survey of trajectory simplification algorithms, see [Sun et al., 2016; Shi and Cheung, 2006; Feng and Shen, 2017]. In an online environment, global algorithms are not very useful. Online trajectory simplification techniques have been explored by [Trajcevski et al., 2006; Lange et al., 2008; Liu et al., 2012, 2013; Muckell et al., 2011; Meng et al., 2017].

Trajectories are captured in a spatial context. The following authors have considered trajectory simplification constrained by a road network using map-matching or snapping to the underlying network: [Cao and Wolfson, 2005; Kellaris et al., 2009; Song et al., 2014; Gotsman and Kanza, 2015; Popa et al., 2015].

Our focus in this paper is to solve the min-# [Imai and Iri, 1988; de Berg et al., 1998] problem without displacing original sampled instants or introducing new ones. To the best of our knowledge, very little research has focused on solving the self-intersection problem in trajectories. Constrained or unconstrained moving objects often generate complex linear geometries; a trajectory can have circular loops or self-crossing patterns. For example, interconnected overpasses of a road network will lead to paths that cross in a non-planar fashion; movement of tracked animals or vessels at sea sometimes have self-crossings.

Most simplification algorithms in the literature consider trajectories as simple polylines and do not handle non-planar loop patterns or self-crossing paths. Furthermore, most trajectories and movement patterns in the physical world are constrained and share a spatial relationship to other objects in the same planar space. As shown in Fig. 4.1, the semantic representation of a trajectory irrespective of the simplification algorithm is lost if objects that constrain the spatiotemporal characteristics of the trajectory are not considered during simplification.

## 4.4 Algorithmic Formulation

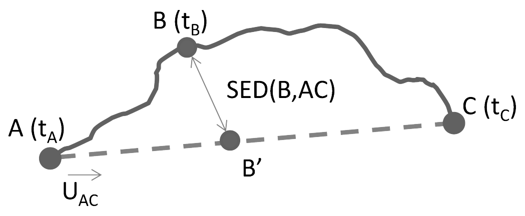
In this paper, we focus on research limitations mentioned in Section 4.3 in an online streaming environment using open window time ratio simplification algorithm [Keogh et al., 2001; Meratnia and Rolf, 2004]. We extend the work of [Tienaah et al., 2015; Stefanakis, 2012] and Tienaah et al. [2018] by external (I/O) processing of contextual trajectory simplification. Tienaah et al. [2018] focused on batch mode, in-memory constrained simplification as applied to polylines without any spatio temporal component. In this paper we focus on external memory (storage), online-simplification with spatiotemporal trajectories constrained by planar objects (points, lines, and polygons).

### 4.4.1 Sampling Algorithm

The SED sampling algorithm [Meratnia and Rolf, 2004] is an adaptation of RDP [Ramer, 1972; Douglas and Peucker, 1973] by changing the distance error function. Given an error threshold $T$ and a polyline $P$ with an ordered set of $n + 1$ vertices $V_0, V_1, V_i, ..., V_n$, the RDP algorithm performs a recursive simplification of $P$ by keeping a generalised segment $\overline{V_0V_n}$ fixed. It finds a splitting vertex $V_k$ by computing a distance offset of intermediate vertices $(V_1, ..., V_{n-1})$ from $\overline{V_0V_n}$. $V_k$ is the vertex with maximum $(T_k)$ of the minimum distances (Hausdorff distance) from $\overline{V_0V_n}$. If $T_k > T$, split the input polyline at $V_k$ with sub-polylines $P_{0-k}$ and $P_{k-n}$. RDP recursively simplifies $P_{0-k}$ and $P_{k-n}$ based on $T$; the recursion terminates if $T_k \leq T$ or $P_{i-j}$ is a segment. The complexity of RDP is $\mathcal{O}(n^2)$. The RDP algorithm has been improved using the path-hull method to $\mathcal{O}(n \log n)$ by Hershberger and Snoeyink [1992] for simple polylines; this improved implementation is not suited for arbitrary polylines.

#### 4.4.1.1 SED Algorithm

The SED algorithm follows the top down recursive simplification of RDP except for how the error metric (threshold $T$ from section 4.4.1) is calculated (see Algorithm 4.2). In a trajectory, each spatio temporal instant $V_i$ can be expressed as a tuple of three coordinates $(V_x, V_y, V_t)$, $V_t$ indicates the time a moving object was at $V_x, V_y$. In figure 4.2, $V_A$, $V_B$, and $V_C$ are three spatiotemporal instants for trajectory $M$, where $t_A < t_B < t_C$ (Figure 4.2). The SED for point $B$ is equal to the Euclidean distance $BB'$, where the location of $B'$ is the position $B$ on the generalized segment (dash) $\overline{AC}$ with respect to the velocity vector $U_{AC}$ (Figure 4.2b). In other words, the offset ($T_B$) of $B$ is the distance from point $B$ to $B'$, where $B'$ is the spatio-temporal trace of $B$ on to $\overline{AC}$ at time $t_B$ [Meratnia and Rolf, 2004; Tienaah et al., 2015].



(a) Trajectory

$$U_A C = \frac{AC}{t_C - t_A} \qquad (4.1)$$
$$AB' = U_{AC}(t_B - t_A)$$

(b) Trace of $B$ as $B'$ on $U_{AC}$

Figure 4.2: Synchronized Euclidean Distance

**Algorithm 4.2** Synchronized Euclidean Distance Offset

```
1:  function SED($coordinates, i, j$)        ▷ coordinates [(x, y, time),...], indices: i, j
2:      $n \leftarrow$ length(coordinates)
3:      $\overline{V_iV_j} \leftarrow$ segment($coordinates, i, j$)
4:      $k, \varepsilon_k \leftarrow j, 0$        ▷ defaults: index $j$ and $\varepsilon = 0.0$
5:      if $j - i > 1$ then
6:          for $c$ in $coordinates[i+1...j-1]$ do        ▷ exclude end points at $i$ and $j$
7:              $V_{sed} \leftarrow$ SEDVector( $\overline{V_iV_j}, coordinates[c]$ )
8:              $\varepsilon \leftarrow$ magnitude($V_{sed}$)
9:              if $\varepsilon \geq \varepsilon_k$ then
10:                 $k, \varepsilon_k \leftarrow c, \varepsilon$
11:     return $k, \varepsilon_k$
```

#### 4.4.1.2 Opening window algorithm

Opening or sliding window (OPW) is an online algorithm [Keogh et al., 2001; Meratnia and Rolf, 2004]. Given an ordered stream of spatiotemporal instants $V_i, V_{i+1}, V_{i+2}, ..., V_{i+n}$ of a moving object, OPW starts by creating a segment $\overline{V_i V_{i+2}}$ between $V_i$ (anchor) and $V_{i+2}$ (floater) in an ordered series of incoming spatiotemporal instants. It then computes the distance offset $T_{i+1}$ from $\overline{V_i V_{i+2}}$, if $T_{i+1} < T$, a new generalized segment is created by extending the floater to $V_{i+3}$. The offset computation is repeated for all intermediate instants between the anchor and floater. The vertex $V_{i+k}$ with $T_k > T$ becomes the stop condition with two strategies:

- Normal Opening Window (NOPW) - split the stream at $V_{i+k}$ and the sub-stream from $V_i$ to $V_{i+k}$ can be generalized as $\overline{V_i V_{i+k}}$; move the anchor to $V_{i+k}$ and floater to $V_n$.
- Before Opening Window (BOPW) - split the stream at $V_{n-1}$ and the sub-stream from $V_i$ to $V_{n-1}$ can be generalized as $\overline{V_i V_{n-1}}$; move the anchor to $V_{n-1}$ and floater to $V_n$.

In this paper, we focus on NOPW using SED as error function. The complexity of node aggregation in memory using NOPW is $\mathcal{O}(n^2)$, where $n$ is the number of spatiotemporal instants in cache.

## 4.5 Implementation

Consider a monitoring station $S$ listening for spatiotemporal instants of $N$ vessels at sea; $S$ has a temporal memory $B$ that can store $C$ number of instants per vessel in $N$, $C = B/N$. $C$ is the maximum cache size for each vessel before the memory is flushed to a secondary storage. In this paper, we assume an infinite external secondary storage for a trajectory stream. We store all instants of the stream from $V_i$ to $V_j$ as node $N_{i,j}$, where $i < j$. We use intermediate instants (between $i$ and $j$) for resolving inconsistent simplification of $N_{i,j}$.

The field properties of $N_{i,j}$ are as follows:

```
struct Node {
    $fid$           : $int$                            ▷ feature id
    $nid$           : $int$                            ▷ node id
    $coordinates$   : $Array[Point3d]$                 ▷ coordinate array of 3d points (x, y, time)
    $range$         : $[int, int]$                     ▷ indices along stream (anchor, floater): i, j
}
```

The structure of $N_{i,j}$ is represented in an external database as:

```
Node Table (
    $id$        : $serial$      NOT NULL           ▷ constraint: primary key
    $fid$       : $int$         NOT NULL
    $node$      : $text$        NOT NULL
    $geom$      : $geometry$    NOT NULL
    $i$         : $int$         NOT NULL
    $j$         : $int$         NOT NULL           ▷ constraint: unique (fid, i, j)
    $size$      : $int$         CHECK($size > 0$)
    $status$    : $int$         DEFAULT 0
    $snapshot$  : $int$         DEFAULT 0
)
```

$id$ is the primary key for a trajectory decomposition table, $fid$ represent the feature id for a trajectory. The $size$ of $N_{i,j}$ is computed as $j - i$ where $j > i$, $i$ and $j$ are the indices of $V_i$ and $V_j$ in an ordered instant stream. $size$ is constrained to be greater than 0. $N_{i,j}$ is serialized as text and stored in the field $node$. The convex hull of vertices from $V_i$ to $V_j$ is stored as a: (i) polyline: $V_i, V_k, ..., V_j$ are collinear or a segment when $j - i = 1$ or (ii) polygon: when the area of the convex hull of vertices from $i$ to $j$ is not zero. $status$ represents three states of $N_{i,j}$: $NullState$, $Collapsible$ and $SplitNode$ with values 0, 1 and 2 respectively. $NullState$ is the default state indicating the node has not been checked for constraints; $Collapsible$ indicates $N_{i,j}$ can be represented as a spatiotemporal segment $\overline{V_iV_j}$; and $SplitNode$ indicates $N_{i,j}$ require further processing to be consistent. $snapshot$ is a flag used to indicate which nodes are being simplified while online streaming. $snapshot$ has two states: $UnSnap$ and $Snap$ with values 0 and 1 respectively; the default $snap$ state of a $N_{i,j}$ is $UnSnap$.

A node table is populated by the fields and derived field values of $N_{i,j}$. $N_{i,j}$ serves as a context collapsible unit of a given moving object with a unique $fid$. To speed up processing, node table fields: $fid$, $i$, $j$, $size$, $status$, $snapshot$ are indexed using a *B-Tree* and $geometry$ as a PostgreSQL/PostGIS generic index structure *GIST*.

Figure 4.3 shows a layout of online trajectory processing: input stream, aggregation, simplification, and storage in an external database. Constrained simplification of nodes are done by taking into account contextual objects within its neighbourhood of influence. We discuss how to resolve simplification errors in section 4.5.1.

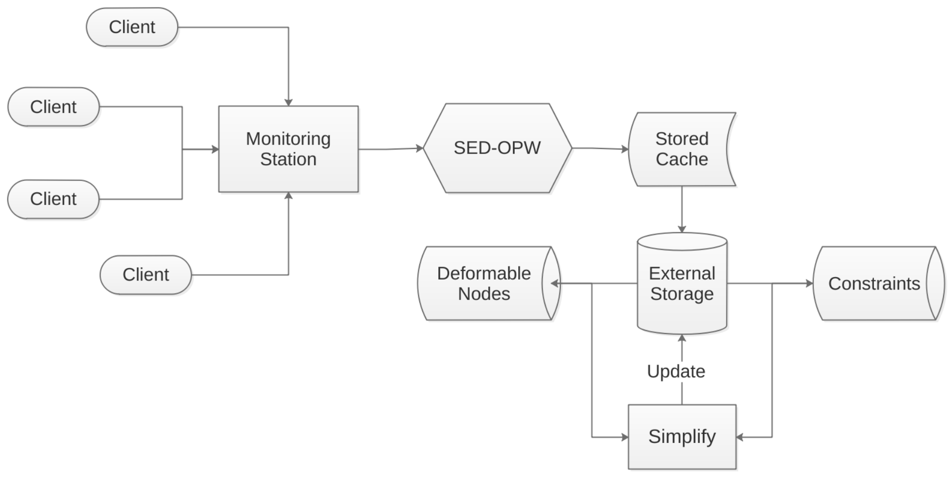


Figure 4.3: Online processing of trajectory streams

The spatiotemporal collapse of $N_{i,j}$ to form a chain of piecewise approximated segment ($\overline{V_iV_j}$) requires spatial relations of the original trajectory to other objects that will change spatial meaning as a result of approximation. Each node is considered “collapsible” if its segment preserves the spatial relations of the original trajectory, otherwise it is considered to be “deformable”. To make deformable nodes collapsible, we proceed by finding $V_k$ using the SED algorithm and split $N_{i,j}$ at $k$ to form two new nodes: $N_{i,k}$ and $N_{k,j}$. The database is updated by inserting $N_{i,k}$, $N_{k,j}$ and removing $N_{i,j}$. This iterative process continuous until all the nodes are collapsible (see Algorithm 4.3).

**Algorithm 4.3** Constrained Trajectory Simplification

```
1: function SIMPLIFY($fid$ : $int$)                    ▷ simplification of $fid$ at a $snapshot$
2:     markSnapshot($fid$, $Snap$)
3:     while <has more deformables for $fid$> do
4:         markDeformables($fid$)
5:         markNullStateAsCollapsible($fid$)
6:         splitDeformables($fid$)
7:         cleanUpDeformables($fid$)
8:     aggregateSimpleSegments($fid$, $MergeFragmentSize$)
9:     saveSimplification($fid$)
10:    markSnapshot($fid$, $UnSnap$)
```

Our implementation concurrently performs SED-OPW aggregation and constrained simplification of stored nodes of multiple trajectories. `Simplify` (Algorithm 4.3) starts by first setting the $snapshot$ field of a given $fid$ to $Snap$ ($SnapNodes$). This flag differentiates nodes been simplified and newly saved nodes from the concurrent SED-OPW process. While there are nodes of $fid$ with $snapshot = Snap$ and $status = NullState$, constrain $SnapNodes$ by context neighbours and simplification options; set $status$ field of deformable nodes as $SplitNode$. Update rest of the $SnapNodes$ as $Collapsible$. The next step in simplification is to split and update the nodes table of all $SnapNodes$ with $status = SplitNode$. These newly inserted split nodes have $status = NullState$ and $snapshot = Snap$. The clean-up step in `Simplify` deletes all deformable $SnapNodes$ nodes from the nodes table. While there are nodes for $fid$ with $status = NullState$ and $snapshot = Snap$ the process iterates until all $SnapNodes$ are collapsible.

After the main loop of `Simplify`, we seek to merge all contiguous $SnapNodes$ for $fid$ that can be merged based on a simplification error threshold. Contiguous $SnapNodes$ that can be merged are deleted from the nodes table and replaced with a new node that encompasses their ordered instants from $i$ to $j$. A temporal simplification state of $fid$ is approximated as a contiguous chain of collapsible $SnapNodes$; this temporal simplification is saved in the $geometry$ field of an output table. This is useful in an online environment, where the correct constrained simplification of $fid$ can be queried while object is in motion.

Finally, `Simplify` completes a simplification snapshot by setting the $snapshot$ flag for $SnapNodes$ to $UnSnap$. Our concurrent simplification process resumes a new batch of trajectory simplification after idling for one second. The simplification process will be idle when the $status$ of all the nodes are set as $Collapsible$.

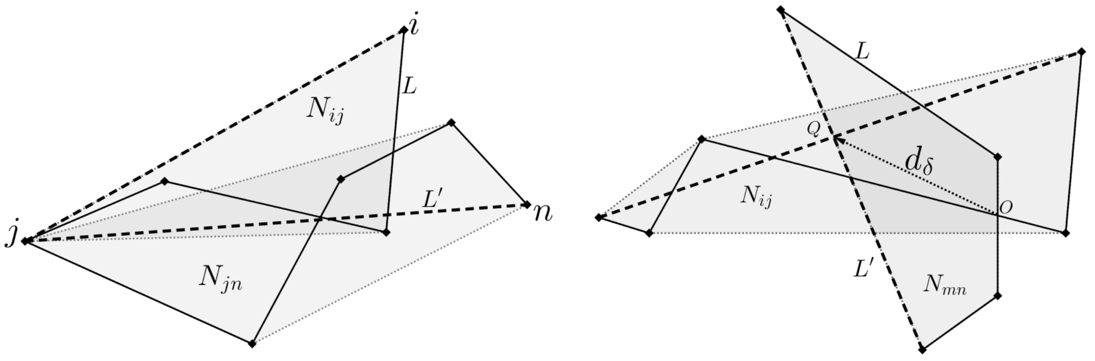
### 4.5.1 Constrained Simplification

In this paper we consider three spatial relations to be observed by a moving vessels at sea in the context of islands as constraints: (i) geometric intersect/disjoint relation, (ii) homotopy of simplified trajectory, (iii) minimum distance to islands. We also consider two topological properties of a trajectory: (i) self-intersection as a result of simplification, and (ii) non-planar intersection between simplification units ($N_{i,j}$, $N_{m,n}$). We ignore self-intersections within aggregation units since their simplification error is at most $\varepsilon$. We also limit our scope to topological relations between aggregation units of the same trajectory and not between trajectories. Using the spatio temporal aggregation by SED-OPW algorithm, $N_{i,j}$ is collapsible as $\overline{V_iV_j}$ if it preserves these spatial and topological relations in the context of islands. If $N_{i,j}$ is not collapsible, it is marked as deformable as described in Algorithm 4.3. We consider each constraint in detail.

#### 4.5.1.1 Topological Relations

Consecutive and non-consecutive $N_{i,j}$ units can have non-planar self-intersection (see Fig. 4.4). We also constrain the non-planar relationship between aggregation units using a simple geometric heuristic 4.1.

**Heuristic 4.1.** *Deform $N_{i,j}$ and $N_{j,n}$, if $\overline{V_iV_j}$ and $\overline{V_jV_n}$ intersect only at $j$ but $N_{i,j}$ and $N_{j,n}$ intersect other than $j$, see Fig. 4.4a.*



(a) contiguous (b) non-contiguous

Figure 4.4: (a) Contiguous aggregation units with non-planar intersects (b) overlap of non-contiguous aggregation units

An example of a situation containing non-planar crossings between non-contiguous aggregation units of the same trajectory is illustrated in Fig. 4.4b. We constrain the deviation of the intersection between simplified segments by a "relax" distance ($\delta$) - the offset between the intersect of sub-polylines and that of their generalized segments.

**Heuristic 4.2.** *Deform $N_{i,j}$ and $N_{m,n}$, where $j \neq m$ and $n \neq i$, if the distance ($d_\delta$) between $O$ and $Q$ is greater than $\delta$, where $O$ and $Q$ is the intersect between $N_{i,j}$,$N_{m,n}$ and segments $\overline{V_iV_j}$,$\overline{V_mV_n}$ respectively. See Fig. 4.4b.*

To avoid self-intersection introduced by the simplification algorithm, we use heuristic 4.3. Two aggregation units can avoid self-intersection if their sub-polylines are disjoint and the intersect between corresponding generalized segments are also disjoint (Fig. 4.5b). An intersection between two generalized segments of non-consecutive aggregation units is inconsistent if their sub-polylines are disjoint (Fig 4.5a). Using heuristic 4.3, we decide between which aggregation unit to further deform using the SED algorithm.

**Heuristic 4.3.** *Given two intersecting non-consecutive aggregation units $N_{i,j}$ and $N_{m,n}$ with disjoint sub-polylines, deform $N_{i,j}$, if $\overline{V_iV_j}$ intersects $N_{m,n}$ and $\overline{V_mV_n}$.*

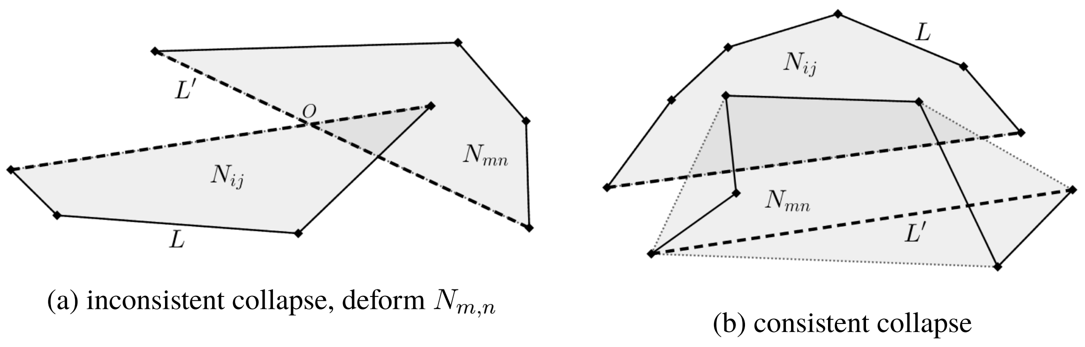


Figure 4.5: (a) Inconsistent collapse of $N_{m,n}$ based on heuristic 4.3 (b) disjoint sub-polylines with consistent collapsed segments (disjoint)

#### 4.5.1.2 Geometric Relation

We consider the intersection and disjoint relation of a trajectory and its simplified spatiotemporal segment to an island, see Figure 4.6. $\overline{V_iV_j}$ is consistent if it has the same geometric relation as $N_{i,j}$ to a planar object.

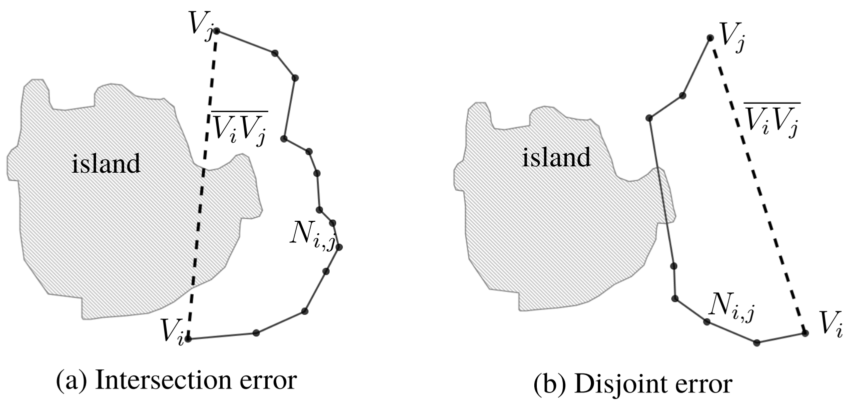


Figure 4.6: (a) Trajectory aggregate $N_{i,j}$ (solid polyline) is disjoint but $\overline{V_iV_j}$ has an intersect relation to the island. (b)Trajectory aggregate $N_{i,j}$ (solid polyline) intersects but $\overline{V_iV_j}$ has an disjoint relation to the island. In (b), due to transmission issues or sampling interval, the trajectory reconstruction can intersect an island. (a) and (b) are not collapsible.

#### 4.5.1.3 Proximity Relation

A minimum distance constraint requires $\overline{V_iV_j}$ be within a given $\delta$ distance of other planar objects. See Fig. 4.7. If the original sub-polyline ($N_{i,j}$) violates the minimum distance constraint, $\overline{V_iV_j}$ is considered to be a valid collapse.

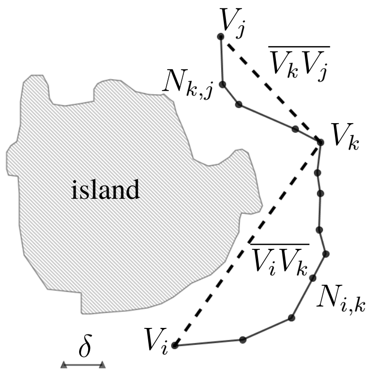


Figure 4.7: Minimum distance relation to a planar object. The minimum distance from $\overline{V_iV_k}$ to the island is less than $\delta$; $N_{i,k}$ requires further decomposition using SED. $\overline{V_kV_j}$ is a valid collapse - has a bigger proximity to the island compared to $N_{k,j}$

### 4.5.1.4 Homotopy Relation

The homotopy relation captures the spatial collapsibility of $N_{i,j}$ in relation to other planar geometric objects. The sub-polyline $N_{i,j}$ is a boundary that subdivides a planar space into two faces; an object $O_i$ on one face of $N_{i,j}$ should be on the same face as its collapsed segment $\overline{V_iV_j}$. If $O_i$ changes a face (sides), then $\overline{V_iV_j}$ is an inconsistent collapse of $N_{i,j}$. In this paper, we consider $O_i$ to not change face if it intersects both $N_{i,j}$ and $\overline{V_iV_j}$ (Fig. 4.8).

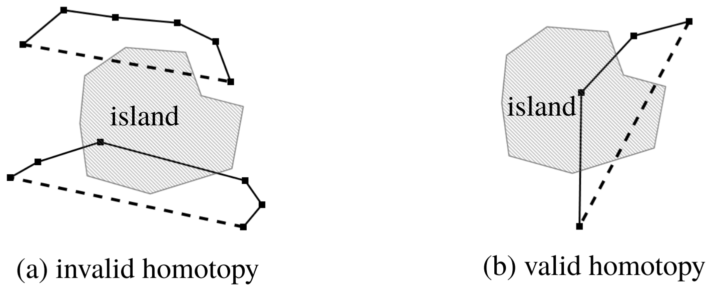
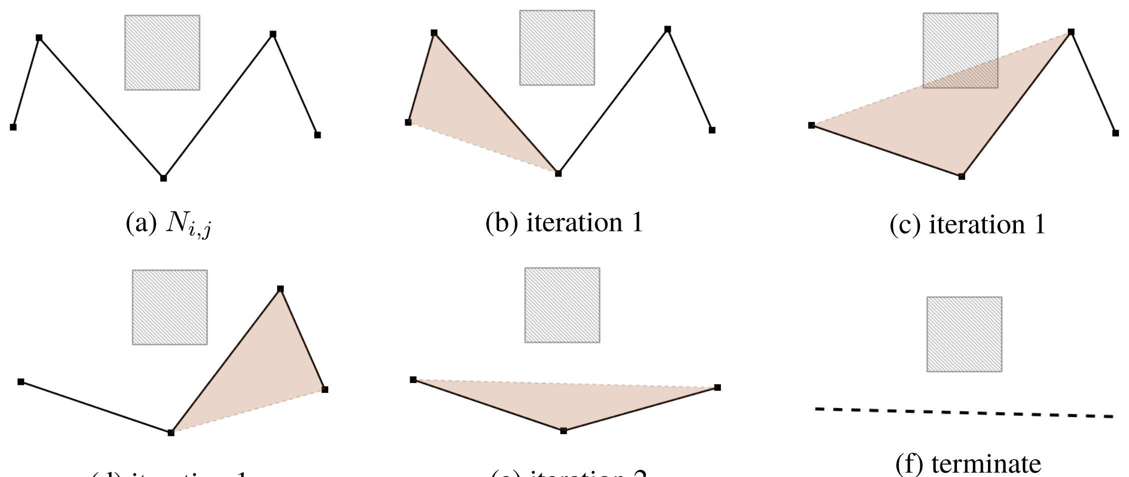


Figure 4.8: (a) Trajectory aggregate (solid) polylines have different homotopy to their simplification (dash segment) (b) Intersecting trajectory aggregate and collapsed segment (dash) are considered to be of the same homotopy class.

To fully capture the collapsibility of $N_{i,j}$ with respect to disjoint context objects, we adopt the chain deformation algorithm by Tienaah et al. [2018]. The algorithm starts by first filtering context objects that are disjoint to a sub-polyline. Instances indicated in Fig. 4.8b are classified as valid. Keeping $V_i$ and $V_j$ of $N_{i,j}$ fixed, we deform $N_{i,j}$

by forming triangles between each intermediate vertex and its immediate neighbouring vertices (previous, current, and next). For example, given a triangle $T_{abc}$, vertex $V_b$ is the current vertex, $V_a$ and $V_c$ are previous and next vertices respectively. We remove the current (middle) vertex if the triangular face is disjoint from context objects. If a triangular face intersects a planar object, that triangular face cannot be collapsed as a line segment (a line from previous to next vertex). The "current" vertex of a triangular face intersecting a planar object is preserved for future iterations. The iterator moves to the "next" vertex as the new "current" vertex and the process (triangular face collapse) is repeated up to $V_{j-1}$. The chain deformation is repeated starting from $V_i$ if the next vertex after $V_i$ is not $V_j$ and at least one triangular collapse occurred in the previous iteration. The deformation algorithm is depicted pictorially in Fig. 4.9. $\overline{V_iV_j}$ is homotopic to $N_{i,j}$ if there are no intermediate vertices between $V_i$ and $V_j$.

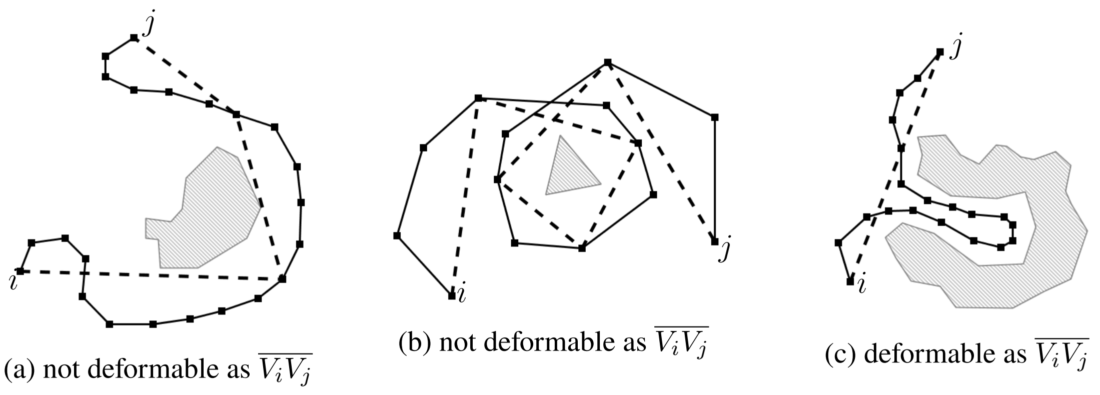


(a) $N_{i,j}$ (b) iteration 1 (c) iteration 1

(d) iteration 1 (e) iteration 2 (f) terminate

Figure 4.9: Homotopic chain deformation

To speed up processing, all disjoint context objects that intersect the the convex hull of $N_{i,j}$ are indexed in an in-memory R-Tree. The complexity of computing the local homotopy $N_{i,j}$ to $\overline{V_iV_j}$ is $\mathcal{O}(n^2)$.



$j$
$i$
(a) not deformable as $\overline{V_iV_j}$
$i$
$j$
(b) not deformable as $\overline{V_iV_j}$
$j$
$i$
(c) deformable as $\overline{V_iV_j}$

Figure 4.10: Homotopic collapse of $N_{i,j}$, $\overline{V_iV_j}$ is homotopic to $N_{i,j}$ in (c) and not homotopic (a) and (b)

A chain of collapsed homotopic $N_{i,j}$ is “strongly homotopic” [Abam et al., 2014], that is, every simplified shortcut $\overline{V_iV_j}$ is homotopic to $N_{i,j}$. A strongly homotopic chain is homotopic to the original trajectory, but not all homotopic chains are strongly homotopic. Figure 4.11 contains four nodes ($N_{0,3}$, $N_{3,8}$, $N_{8,13}$, $N_{13,20}$) and context object $O_1$. $\overline{V_3V_8}$ and $\overline{V_{13}V_{20}}$ are not homotopic to $N_{3,8}$ and $N_{13,20}$ respectively. $L'$ (dash) is homotopic to $L$ (solid) but not *strongly* homotopic.

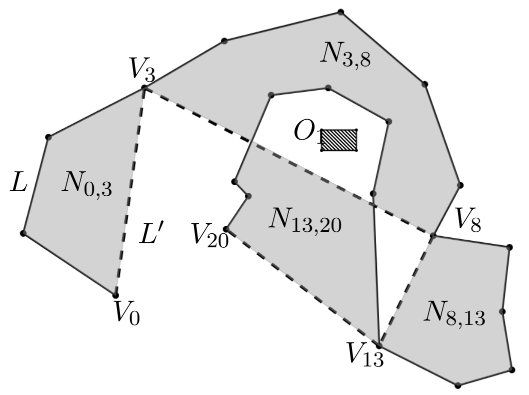


Figure 4.11: Homotopy of a simplification in the context of a planar object. $L'$ (dash polyline) is homotopic to $L$ (solid polyline) but not strongly homotopic

## 4.6 Experimental Evaluation

To evaluate the correctness and performance of our implementation, we use one hundred (100) trajectories from Marine Traffic (*www.marinetraffic.com*) archive (2012) of moving vessels in the Aegean. The dataset consists of $33,842$ temporal instants covering a distance of $68,976.48km$ ($4,134.17$ hours of movement). Each trajectory is simplified at an SED offset ($\varepsilon_{SED}$) from $0.5km$ to $50km$ at intervals of $0.5km$. We constrain simplified trajectories to a minimum distance of $100m$ from islands. Non-planar intersections between aggregation units are constrained to a "relax" distance of $10m$ (see heuristic 4.2). The layout of client, server and database architecture is as shown in Fig. 4.3. Experiments are performed on an Intel® Core™ i7 3.6GHz x4, 16GB RAM. To accelerate processing and queries to a spatial enabled database (PostgreSQL/PostGIS), node table fields: $fid$, $i$, $j$, $size$, $status$, $snapshot$ are indexed using a *B-Tree* and $geometry$ as a generic index structure *GIST*.

Spatio temporal transmissions to the server are in eight (8) concurrent batches. A client-server request and response is processed at both ends in a few milliseconds before the next transmission(actual time interval between recorded vessel instants can be several minutes, e.g., 10 or 20 minutes). The server registers the first transmission and drop subsequent instants tagged as *at anchor*, *moored* and *aground*.

The spatiotemporal instants are aggregated in temporary memory using SED-NOPW at a minimum cache limit of three (3) and maximum of one million ($1,000,000$). This is to make sure $N_{i,j}$ aggregates are done by SED distance at a given $\varepsilon_{SED}$ and not by temporary memory size restriction. The spatio temporal instants in the cache form $N_{i,j}$ where $V_j$ is the vertex at which cache $SED > \varepsilon_{SED}$; $N_{i,j}$ is saved to the nodes table. $V_j$ becomes the first vertex of the new cache.

While transmission from client to server, records in the nodes table are being simplified in batches of eight (8) vessels. Constrained simplification input options include $\varepsilon_{SED}$, self-intersection, non-planar intersection displacement between aggregation units, intersect and disjoint relation, proximity to islands, and homotopy of simplification. In the unconstrained simplification, $\varepsilon_{SED}$ is the only input option required. Data reduction over variable $\varepsilon_{SED}$ is shown in Fig. 4.12. The processing time with respect to change in $\varepsilon_{SED}$ is shown in

Fig. 4.13.

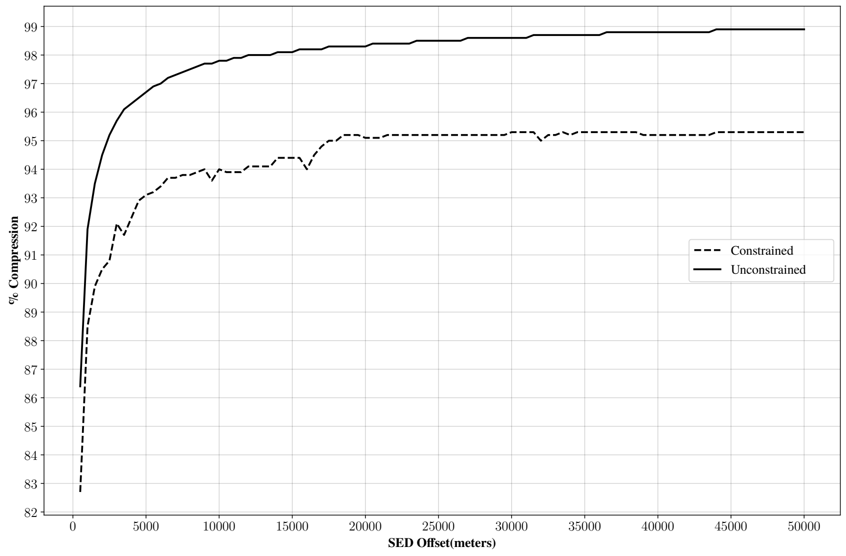


Figure 4.12: Percentage Compression over variable $\varepsilon_{SED}$

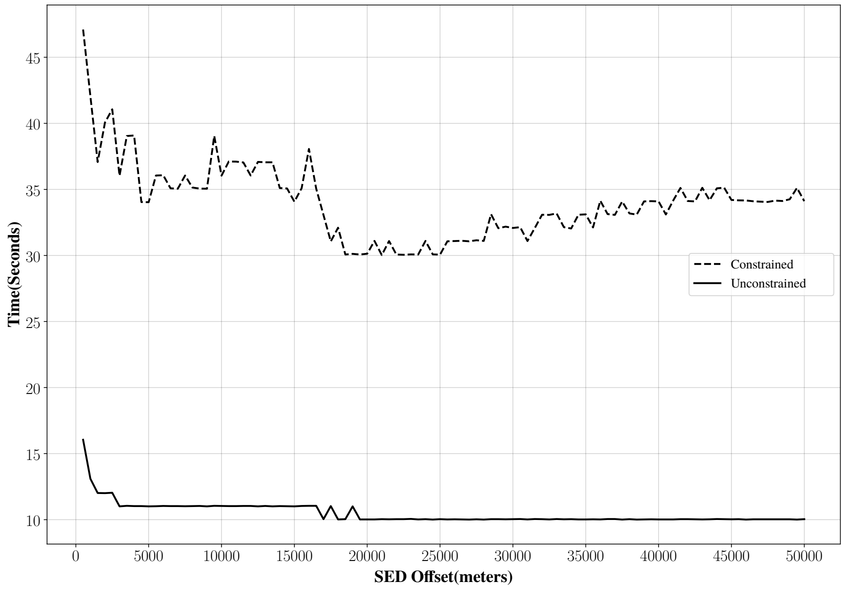


Figure 4.13: Processing Time over variable $\varepsilon_{SED}$

The results in Fig. 4.12 shows a higher compression rate in unconstrained versus constrained simplification. This is expected since, in order to constrain a trajectory in the context of other geometries, we reintroduce some of the vertices discarded by the unconstrained simplification to remain consistent (topology or spatial relation). Figures 4.14, 4.15, 4.16, and 4.17 show a visual comparison between contained and unconstrained trajectory simplification in the context of islands in the Aegean.

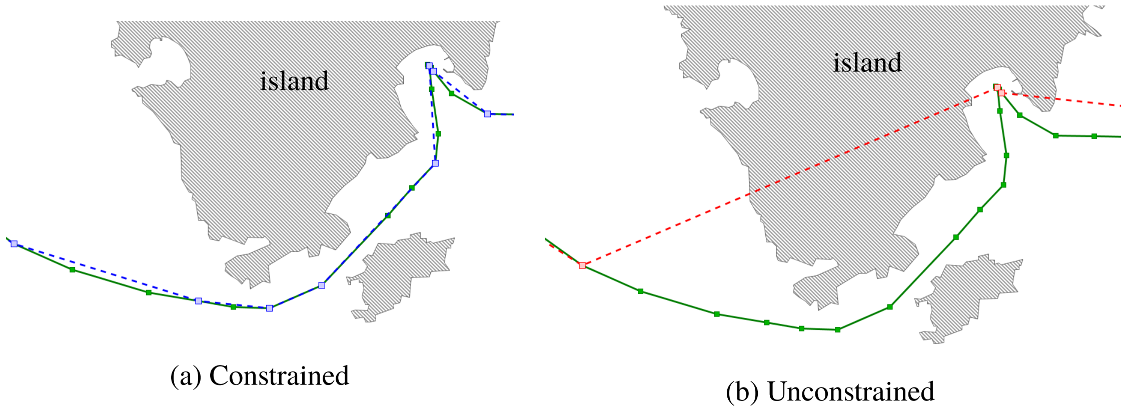


Figure 4.14: Original trajectory (solid) (a) constrained simplification (dashed) maintains geometric relation (b) unconstrained simplification (dashed) intersects island.

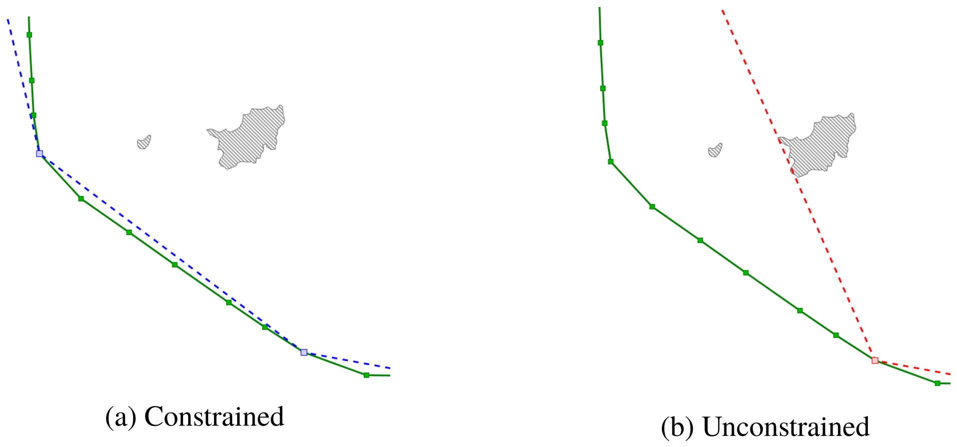


Figure 4.15: Original trajectory (solid) (a) constrained simplification (dashed) maintains geometric, homotopy and minimum distance relation (b) unconstrained simplification (dashed) shows an inconsistent homotopy and spatial relation to island.

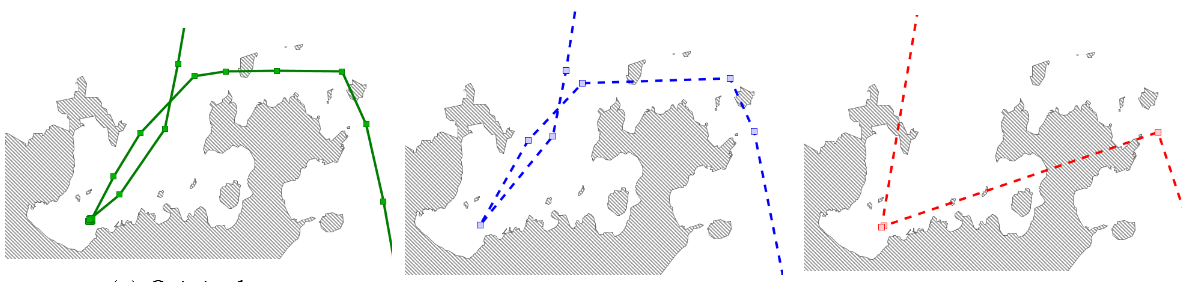

(a) Original (b) Constrained (c) Unconstrained

Figure 4.16: Original trajectory (solid) (b) constrained simplification (dashed) maintains geometric relation in (a); (c) unconstrained simplification (dashed) violates spatial relations in (a).

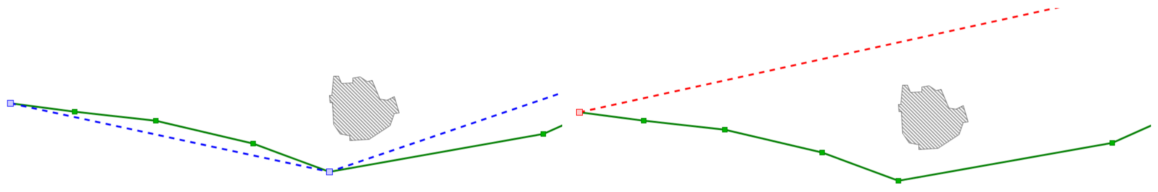

(a) Constrained (b) Unconstrained

Figure 4.17: Original trajectory (solid) (a) constrained simplification (dashed) maintains homotopy relation; (b) unconstrained simplification (dashed) inconsistent homotopy relation to island.

## 4.7 Conclusion

In this paper, we demonstrate the first online trajectory simplification scheme to avoid and preserve non-planar self-intersection between aggregation units, geometric relation (disjoint/intersect), proximity, and homotopic characteristics of trajectory streams in an online environment. Our implementation is designed for arbitrary trajectories with planar objects (points, lines and polygons) as constraints. Experimental evaluation using sea vessel trajectories in the Aegean with islands as constraints was conducted. The results show favourable consistent simplification advantage with constrained simplification at about $4\%$ less than unconstrained. Experimental evaluation (Section 4.6) showed that constrained simplification takes about four times the time to process eight concurrent input streams compared to unconstrained simplification. Unconstrained simplification leads to data errors which invalidates the benefits of data reduction.

Our implementation used an external storage resource and supported real-time queries during on-line aggregation and processing. Future research should focus on efficient implementation to improve running time with hard real-time requirements.

# References

Abam, M. A., Daneshpajouh, S., Deleuran, L., Ehsani, S., and Ghodsi, M. (2014). Computing homotopic line simplification. *Computational Geometry*, 47(7):728–739.

Birnbaum, J., Meng, H.-C., Hwang, J.-H., and Lawson, C. (2013). Similarity-based compression of gps trajectory data. In *Computing for Geospatial Research and Application (COM. Geo), 2013 Fourth International Conference on*, pages 92–95. IEEE.

Cao, H. and Wolfson, O. (2005). Nonmaterialized motion information in transport networks. In *International Conference on Database Theory*, pages 173–188. Springer.

Cao, H., Wolfson, O., and Trajcevski, G. (2006). Spatio-temporal data reduction with deterministic error bounds. *The VLDB Journal—The International Journal on Very Large Data Bases*, 15(3):211–228.

Chen, M., Xu, M., and Franti, P. (2012). A fast $o(n)$ multiresolution polygonal approximation algorithm for gps trajectory simplification. *IEEE Transactions on Image Processing*, 21(5):2770–2785.

Corcoran, P., Mooney, P., and Winstanley, A. (2011). Planar and non-planar topologically consistent vector map simplification. *International Journal of Geographical Information Science*, 25(10):1659–1680.

Daneshpajouh, S. and Ghodsi, M. (2011). A heuristic homotopic path simplification algorithm. In *International Conference on Computational Science and Its Applications*, pages 132–140. Springer.

de Berg, M., van Kreveld, M., and Schirra, S. (1998). Topologically correct subdivision simplification using the bandwidth criterion. *Cartography and Geographic Information Systems*, 25(4):243–257.

Douglas, D. H. and Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122.

Estkowski, R. and Mitchell, J. S. (2001). Simplifying a polygonal subdivision while keeping it simple. In *Proceedings of the seventeenth annual symposium on Computational geometry*, pages 40–49. ACM.

Feng, K. and Shen, Z. (2017). Compressing trajectory for trajectory indexing. In *Proceedings of the 2nd International Conference on Crowd Science and Engineering*, pages 68–71. ACM.

Gotsman, R. and Kanza, Y. (2015). A dilution-matching-encoding compaction of trajectories over road networks. *GeoInformatica*, 19(2):331–364.

Heckbert, P. S. and Garland, M. (1997). Survey of polygonal surface simplification algorithms. Technical report, Carnegie-Mellon Univ. Pittsburgh PA., School of Computer Science.

Hershberger, J. and Snoeyink, J. (1992). Speeding up the Douglas-Peucker line-simplification algorithm. Technical report, Vancouver, BC, Canada, Canada.

Imai, H. and Iri, M. (1988). Polygonal approximations of a curve — formulations and algorithms. In Toussaint, G. T., editor, *Computational Morphology*, volume 6 of *Machine Intelligence and Pattern Recognition*, pages 71–86. North-Holland.

Kellaris, G., Pelekis, N., and Theodoridis, Y. (2009). Trajectory compression under network constraints. In *International Symposium on Spatial and Temporal Databases*, pages 392–398. Springer.

Keogh, E., Chu, S., Hart, D., and Pazzani, M. (2001). An online algorithm for segmenting time series. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 289–296. IEEE.

Lange, R., Dürr, F., and Rothermel, K. (2008). Online trajectory data reduction using connection-preserving dead reckoning. In *proceedings of the 5th annual international conference on mobile and ubiquitous systems: computing, networking, and services*, page 52. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Lin, C.-Y., Hung, C.-C., and Lei, P.-R. (2016). A velocity-preserving trajectory simplification approach. In *Technologies and Applications of Artificial Intelligence (TAAI), 2016 Conference on*, pages 58–65. IEEE.

Liu, G., Iwai, M., and Sezaki, K. (2012). A method for online trajectory simplification by enclosed area metric. *ICMU'12*.

Liu, G., Iwai, M., and Sezaki, K. (2013). An online method for trajectory simplification under uncertainty of gps. *IPSJ Online Transactions*, 6:65–74.

Long, C., Wong, R. C.-W., and Jagadish, H. (2013). Direction-preserving trajectory simplification. *Proceedings of the VLDB Endowment*, 6(10):949–960.

McMaster, R. B. (1987). The geometric properties of numerical generalization. *Geographical Analysis*, 19(4):330–346.

Meng, Q., Yu, X., Yao, C., Li, X., Li, P., and Zhao, X. (2017). Improvement of opw-tr algorithm for compressing gps trajectory data. *Journal of information processing systems*, 13(3):533–545.

Meratnia, N. and Rolf, A. (2004). Spatiotemporal compression techniques for moving point objects. In *International Conference on Extending Database Technology*, pages 765–782. Springer.

Muckell, J., Hwang, J.-H., Patil, V., Lawson, C. T., Ping, F., and Ravi, S. (2011). Squish: an online approach for gps trajectory compression. In *Proceedings of the 2nd International Conference on Computing for Geospatial Research & Applications*, page 13. ACM.

Muckell, J., Olsen, P. W., Hwang, J.-H., Lawson, C. T., and Ravi, S. (2014). Compression of trajectory data: a comprehensive evaluation and new approach. *GeoInformatica*, 18(3):435–460.

Muller, J. (1990). The removal of spatial conflicts in line generalization. *Cartography and Geographic Information Systems*, 17(2):141–149.

Nibali, A. and He, Z. (2015). Trajic: An effective compression system for trajectory data. *IEEE Transactions on Knowledge and Data Engineering*, 27(11):3138–3151.

Popa, I. S., Zeitouni, K., Oria, V., and Kharrat, A. (2015). Spatio-temporal compression of trajectories in road networks. *GeoInformatica*, 19(1):117–145.

Potamias, M., Patroumpas, K., and Sellis, T. (2006). Sampling trajectory streams with spatiotemporal criteria. In *Scientific and Statistical Database Management, 2006. 18th International Conference on*, pages 275–284. IEEE.

Ramer, U. (1972). An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing*, 1(3):244–256.

Saalfeld, A. (1999). Topologically consistent line simplification with the Douglas-Peucker algorithm. *Cartography and Geographic Information Science*, 26(1):7–18.

Shi, W. and Cheung, C. (2006). Performance evaluation of line simplification algorithms for vector generalization. *The Cartographic Journal*, 43(1):27–44.

Song, R., Sun, W., Zheng, B., and Zheng, Y. (2014). Press: A novel framework of trajectory compression in road networks. *Proceedings of the VLDB Endowment*, 7(9):661–672.

Stefanakis, E. (2012). Trajectory generalization under space constraints. In *Proceedings of the 7 th Intl. Conf. on Geographic Information Science (GIScience 2012). Columbus, Ohio*.

Sun, P., Xia, S., Yuan, G., and Li, D. (2016). An overview of moving object trajectory compression algorithms. *Mathematical Problems in Engineering*, 2016.

Tienaah, T., Stefanakis, E., and Coleman, D. (2015). Contextual Douglas-Peucker simplification. *Geomatica*, 69(3):327–338.

Tienaah, T., Stefanakis, E., and Coleman, D. (2018). Line simplification while keeping it simple or complex. In *Line Simplification Under Spatial Constraints*. Geographic Information Systems and Science - Research Lab, Geodesy and Geomatics Engineering, UNB.

Trajcevski, G., Cao, H., Scheuermanny, P., Wolfsonz, O., and Vaccaro, D. (2006). On-line data reduction and the quality of history in moving objects databases. In *Proceedings of the 5th ACM international workshop on Data engineering for wireless and mobile access*, pages 19–26. ACM.

White, E. R. (1985). Assessment of line-generalization algorithms using characteristic points. *The American Cartographer*, 12(1):17–28.

# Chapter 5:

# Extending The Cartographic Toolbox of ESRI ArcGIS

This chapter shows some excerpts from the NSERC Engage project (2015) conducted by the author of this dissertation, Dr. Stefanakis, and in collaboration with Dr. Brent Hall at ESRI Canada. The goal of this project is to create an ArcGIS python Add-In for desktop GIS.

Algorithms developed as part of this dissertation are bundled into an ArcGIS Add-In to make this research practical in industry and academic environments. The Add-In is developed using the *Python* programming language and ArcGIS *ArcPy* package. Users of desktop ArcGIS (revision 10.1 or newer versions) can install and use this Add-In to simplify linear features with other layers as spatial constraints. The Add-In enhances the line generalization toolset in desktop ArcGIS with the following benefits:

(a) an accurate representation of static and dynamic linear features in map layouts and geovisualization,

(b) a sophisticated multi-resolution (multi-scale) representation of the linear features, and

(c) enhanced data preparation processes (e.g., data reduction) for spatial analysis and data mining - spatial relations of the original features are preserved.

First, we present a project layout showing how the packages, modules, documentation and tests are organized.

## 5.1 Introduction

For many applications in cartographic mapping and temporal GIS, it is beneficial to collect data once at a largest scale or highest resolution as a master database. Derived data from the master database can then be used for other applications. Simplification is also a useful pre-processing tool before data mining, graphics visualization, data transmission, and data exploration.

ESRI ArcGIS 10.x ($x \leq 5$) cartographic toolset provides Douglas-Peucker (DP) algorithm for line simplification through the *Point Remove* option of *SimplifyLine_cartography*. In this project, we focus on extending the functionality of line simplification in ArcGIS by providing a context to the DP simplification algorithm for static and spatio-temporal data.

### 5.1.1 Add-In Implementation

The Add-In layout is illustrated in Fig. 5.1. In the $Install/app$ folder, we implement all data structures and algorithms in $linegen$; see $linegen$ layout in Fig. 5.2. In the $linegen$ package, there are two geometry packages: $arcgeom$ and $geom$. $arcgeom$ acts as a wrapper around ArcPy geometry classes, such as : $arcpy.PointGeometry$, $arcpy.Multipoint$, $arcpy.Polyline$, $arcpy.Polygon$ and other ArcPy related utilities. $geom$ consists of modules and packages for $2D$ euclidean geometry. $structs$ in $linegen$ contains some fundamental ($stack$, $deck$, $sset$ - sorted set) and spatial data structures ($rtree$). $linegen$ also contains the core of this project:$constdp$ (constrained DP). The decomposition of polylines and heuristics for a consistent simplification are described in chapter 3.

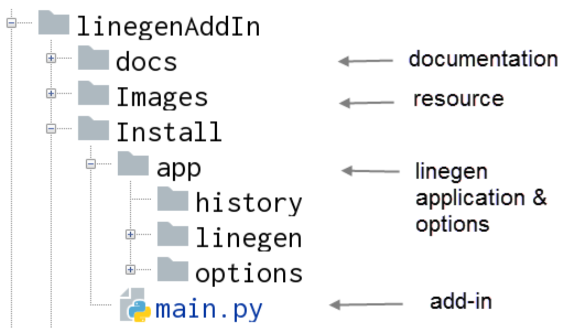
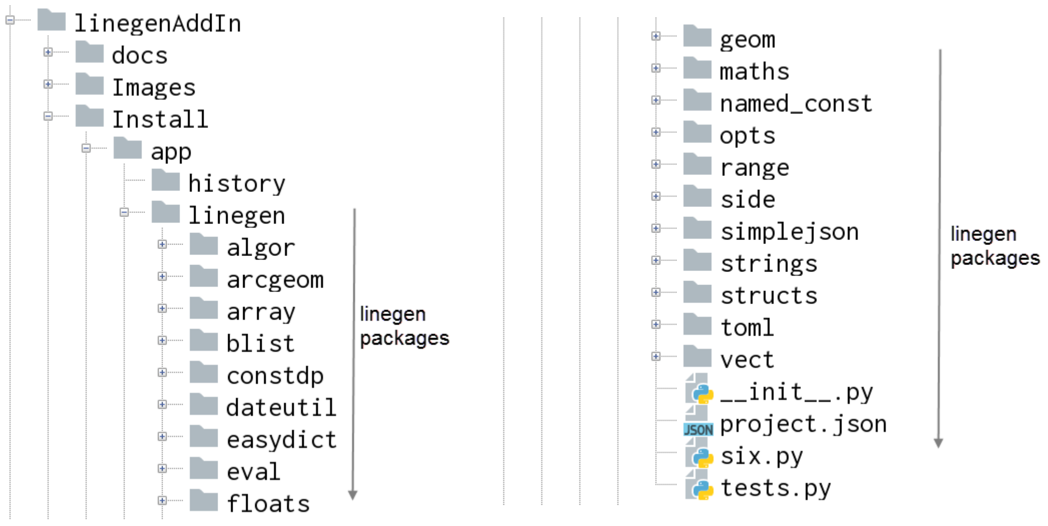


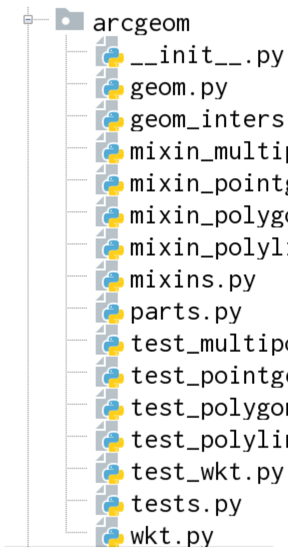
Figure 5.1: Add-In Project Layout



(a) linegen

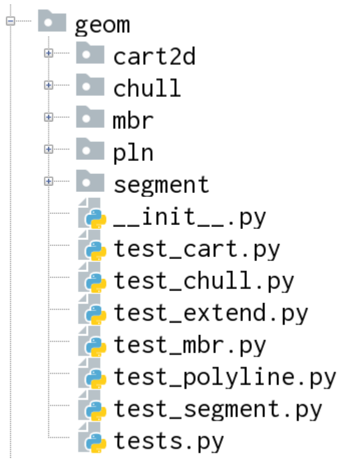
(b) lingen cont.

Figure 5.2: Line Generalization Packages



```
arcgeom
    __init__.py
    geom.py
    geom_inters.py
    mixin_multipoint.py
    mixin_pointgeometry.py
    mixin_polygon.py
    mixin_polyline.py
    mixins.py
    parts.py
    test_multipoint.py
    test_pointgeometry.py
    test_polygon.py
    test_polyline.py
    test_wkt.py
    tests.py
    wkt.py
```

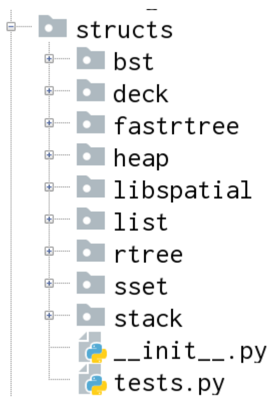
(a) arcgeom



```
geom
    cart2d
    chull
    mbr
    pln
    segment
    __init__.py
    test_cart.py
    test_chull.py
    test_extend.py
    test_mbr.py
    test_polyline.py
    test_segment.py
    tests.py
```

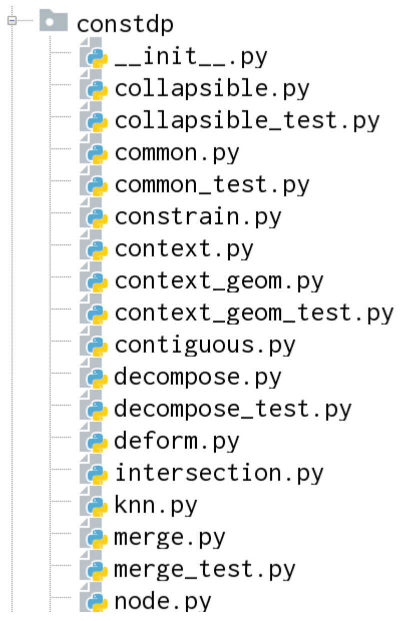
(b) geom

Figure 5.3: Geometry Packages



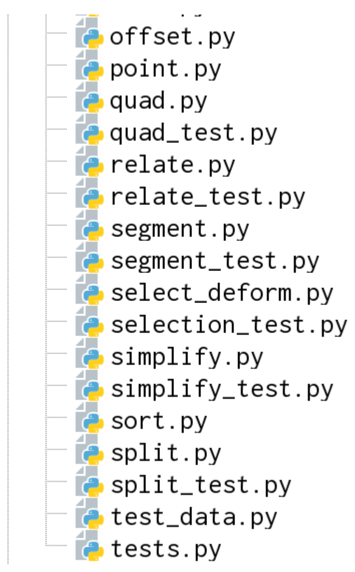
```
structs
    bst
    deck
    fastrtree
    heap
    libspatial
    list
    rtree
    sset
    stack
    __init__.py
    tests.py
```

Figure 5.4: Data Structures



```
constdp
    __init__.py
    collapsible.py
    collapsible_test.py
    common.py
    common_test.py
    constrain.py
    context.py
    context_geom.py
    context_geom_test.py
    contiguous.py
    decompose.py
    decompose_test.py
    deform.py
    intersection.py
    knn.py
    merge.py
    merge_test.py
    node.py
```

(a) constdp



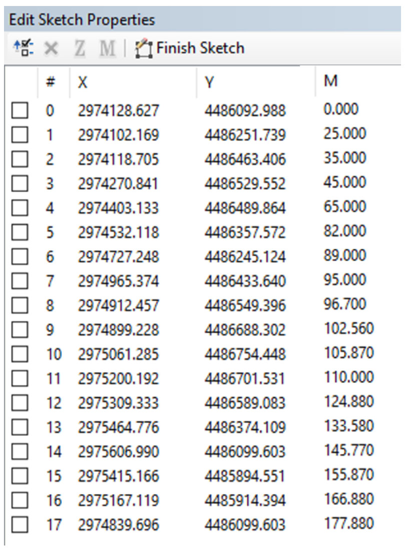
```
    offset.py
    point.py
    quad.py
    quad_test.py
    relate.py
    relate_test.py
    segment.py
    segment_test.py
    select_deform.py
    selection_test.py
    simplify.py
    simplify_test.py
    sort.py
    split.py
    split_test.py
    test_data.py
    tests.py
```

(b) constdp cont.

Figure 5.5: Constrained DP package

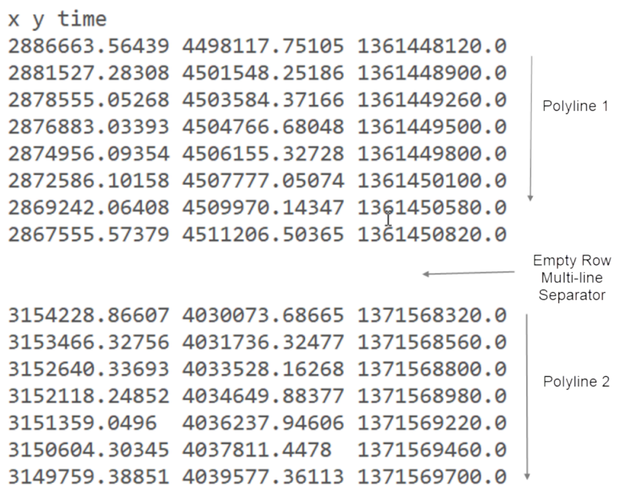
### 5.1.2 Add-In

Two line simplification algorithms are implemented: the Douglas-Peucker (DP) algorithm for static data and Synchronized Euclidean Distance (SED) for spatio-temporal data. The Add-In requires two input data formats: a linear feature class (2D, 3D with Z or M) or a text file with 2/3D coordinates. Constrained and unconstrained DP require at least 2D coordinates and at least 3D for SED simplification. See Figure 5.6.



|  | # | X | Y | M |
|---|---|---|---|---|
| ☐ | 0 | 2974128.627 | 4486092.988 | 0.000 |
| ☐ | 1 | 2974102.169 | 4486251.739 | 25.000 |
| ☐ | 2 | 2974118.705 | 4486463.406 | 35.000 |
| ☐ | 3 | 2974270.841 | 4486529.552 | 45.000 |
| ☐ | 4 | 2974403.133 | 4486489.864 | 65.000 |
| ☐ | 5 | 2974532.118 | 4486357.572 | 82.000 |
| ☐ | 6 | 2974727.248 | 4486245.124 | 89.000 |
| ☐ | 7 | 2974965.374 | 4486433.640 | 95.000 |
| ☐ | 8 | 2974912.457 | 4486549.396 | 96.700 |
| ☐ | 9 | 2974899.228 | 4486688.302 | 102.560 |
| ☐ | 10 | 2975061.285 | 4486754.448 | 105.870 |
| ☐ | 11 | 2975200.192 | 4486701.531 | 110.000 |
| ☐ | 12 | 2975309.333 | 4486589.083 | 124.880 |
| ☐ | 13 | 2975464.776 | 4486374.109 | 133.580 |
| ☐ | 14 | 2975606.990 | 4486099.603 | 145.770 |
| ☐ | 15 | 2975415.166 | 4485894.551 | 155.870 |
| ☐ | 16 | 2975167.119 | 4485914.394 | 166.880 |
| ☐ | 17 | 2974839.696 | 4486099.603 | 177.880 |

(a) Shape properties of linear Feature



```
x y time
2886663.56439 4498117.75105 1361448120.0
2881527.28308 4501548.25186 1361448900.0
2878555.05268 4503584.37166 1361449260.0
2876883.03393 4504766.68048 1361449500.0
2874956.09354 4506155.32728 1361449800.0
2872586.10158 4507777.05074 1361450100.0
2869242.06408 4509970.14347 1361450580.0
2867555.57379 4511206.50365 1361450820.0

3154228.86607 4030073.68665 1371568320.0
3153466.32756 4031736.32477 1371568560.0
3152640.33693 4033528.16268 1371568800.0
3152118.24852 4034649.88377 1371568980.0
3151359.0496  4036237.94606 1371569220.0
3150604.30345 4037811.4478  1371569460.0
3149759.38851 4039577.36113 1371569700.0
```

(b) Text file with multiple polylines (x, y, time(s))

Figure 5.6: Input Data Types

The python Add-In serves as a graphical user interface for constrained and unconstrained DP simplification. Other variants of DP simplification, such as SED benefit from the constrained heuristics of constrained DP. See Add-In toolbar in Fig. 5.7.

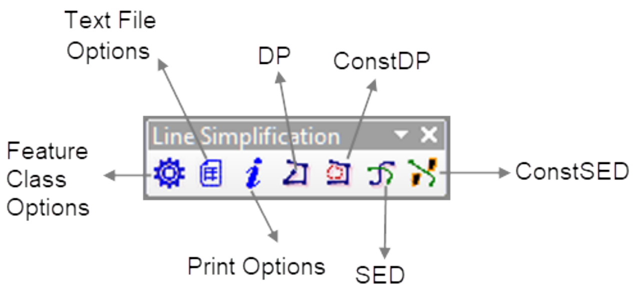
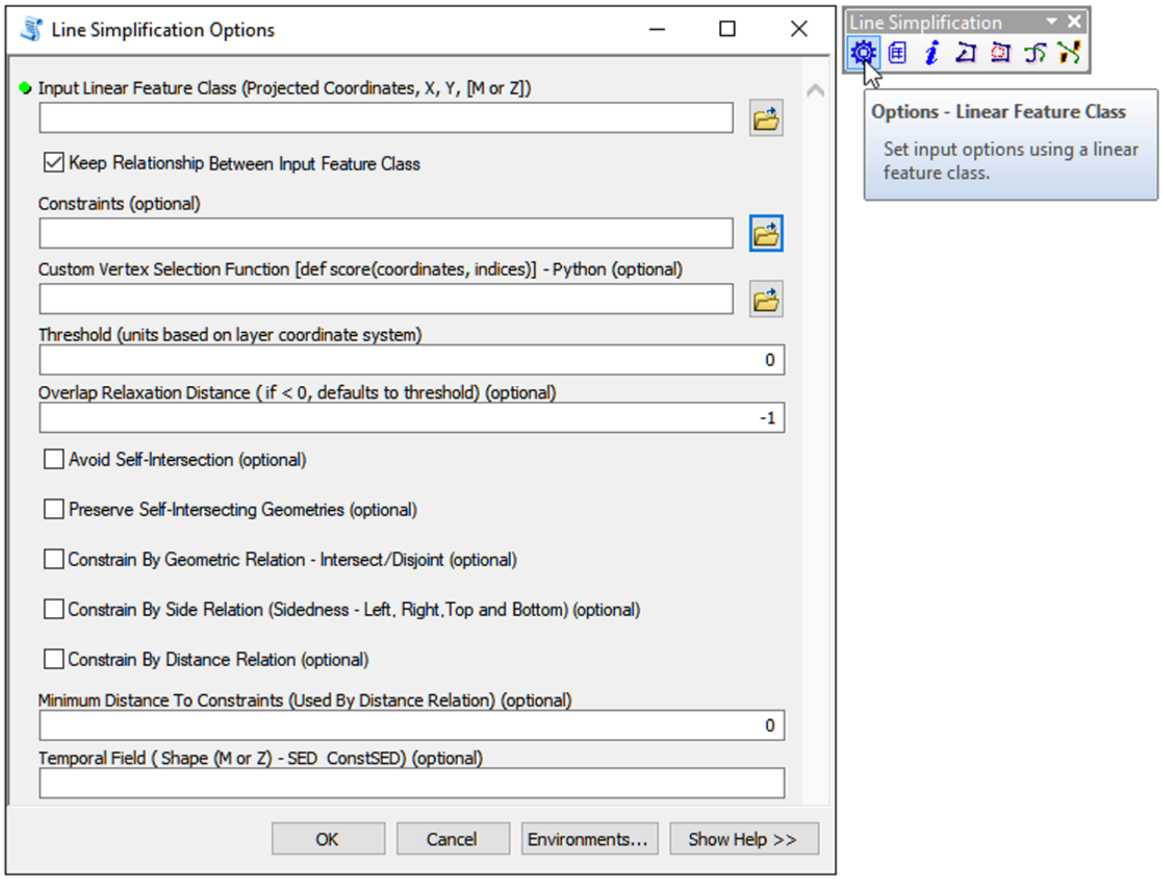


Figure 5.7: Add-In Toolbar

Fig. 5.8 and Fig. 5.9 are graphical interfaces for setting simplification options, see options in Algorithm 0.

### 5.1.3 Input Options: Feature Class

The feature class options is used to set parameters for DP and SED simplification (Figure 5.8). The input feature class can be a polyline for static data or with $M/Z$-values as time for spatiotemporal data.


Line Simplification Options

Input Linear Feature Class (Projected Coordinates, X, Y, [M or Z])

☑ Keep Relationship Between Input Feature Class

Constraints (optional)

Custom Vertex Selection Function [def score(coordinates, indices)] - Python (optional)

Threshold (units based on layer coordinate system): 0

Overlap Relaxation Distance ( if < 0, defaults to threshold) (optional): -1

☐ Avoid Self-Intersection (optional)

☐ Preserve Self-Intersecting Geometries (optional)

☐ Constrain By Geometric Relation - Intersect/Disjoint (optional)

☐ Constrain By Side Relation (Sidedness - Left, Right,Top and Bottom) (optional)

☐ Constrain By Distance Relation (optional)

Minimum Distance To Constraints (Used By Distance Relation) (optional): 0

Temporal Field ( Shape (M or Z) - SED ConstSED) (optional)

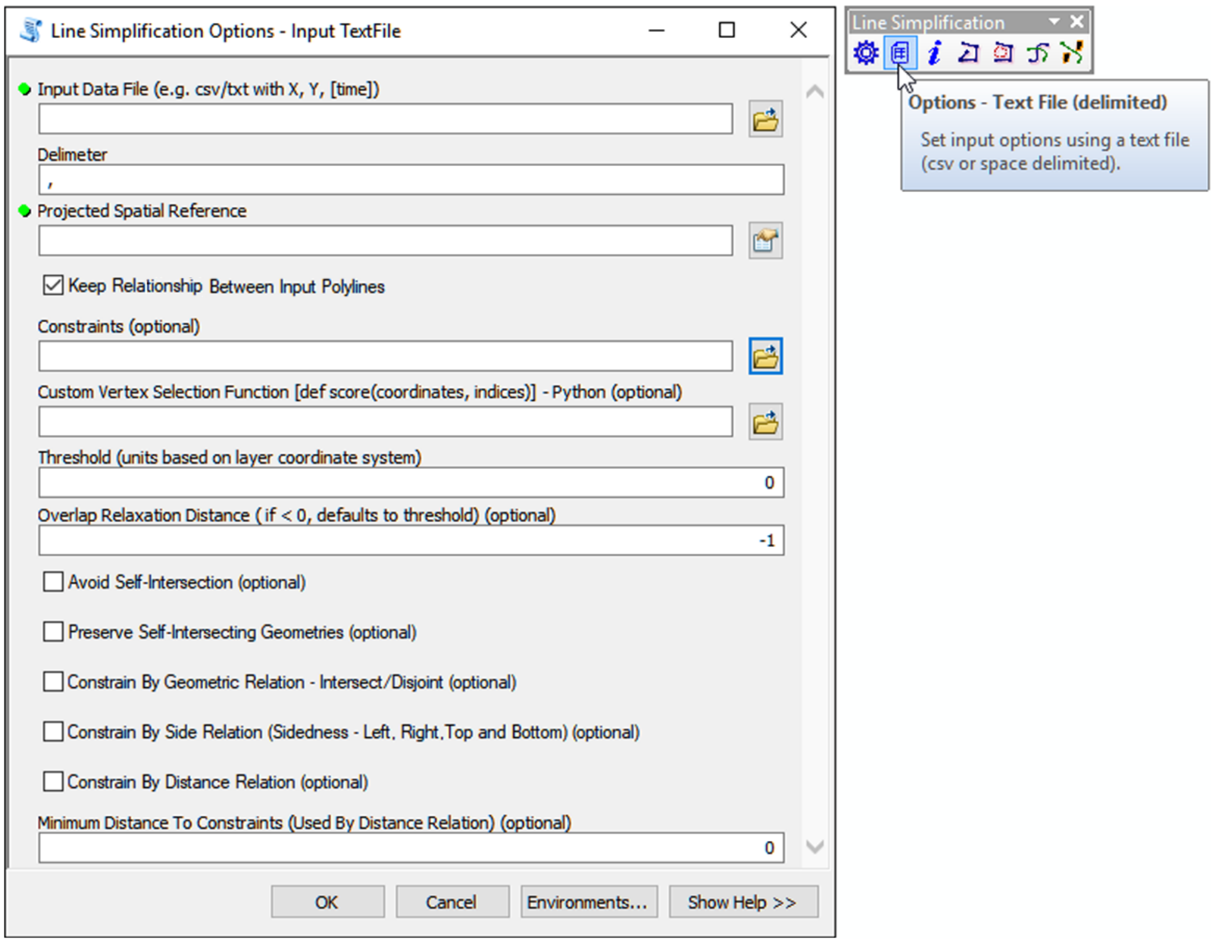
Options - Linear Feature Class

Set input options using a linear feature class.

Figure 5.8: Add-In Feature Class Options

### 5.1.4 Input Options: Text File Options

The text file options is used to set input parameters for the Douglas-Peucker and spatiotemporal simplification using a text file (comma or space separated values) as input (Figure 5.9). A text file designed for SED simplification should have three columns: *X*- and *Y*-coordinates, and a *Time* field. The DP and constrained DP tools can still be used on data loaded using this tool, the time field will be truncated.


Line Simplification Options - Input TextFile

Input Data File (e.g. csv/txt with X, Y, [time])

Delimeter

,

Projected Spatial Reference

☑ Keep Relationship Between Input Polylines

Constraints (optional)

Custom Vertex Selection Function [def score(coordinates, indices)] - Python (optional)

Threshold (units based on layer coordinate system)

0

Overlap Relaxation Distance ( if < 0, defaults to threshold) (optional)

-1

☐ Avoid Self-Intersection (optional)

☐ Preserve Self-Intersecting Geometries (optional)

☐ Constrain By Geometric Relation - Intersect/Disjoint (optional)

☐ Constrain By Side Relation (Sidedness - Left, Right,Top and Bottom) (optional)

☐ Constrain By Distance Relation (optional)

Minimum Distance To Constraints (Used By Distance Relation) (optional)

0

Options - Text File (delimited)

Set input options using a text file (csv or space delimited).

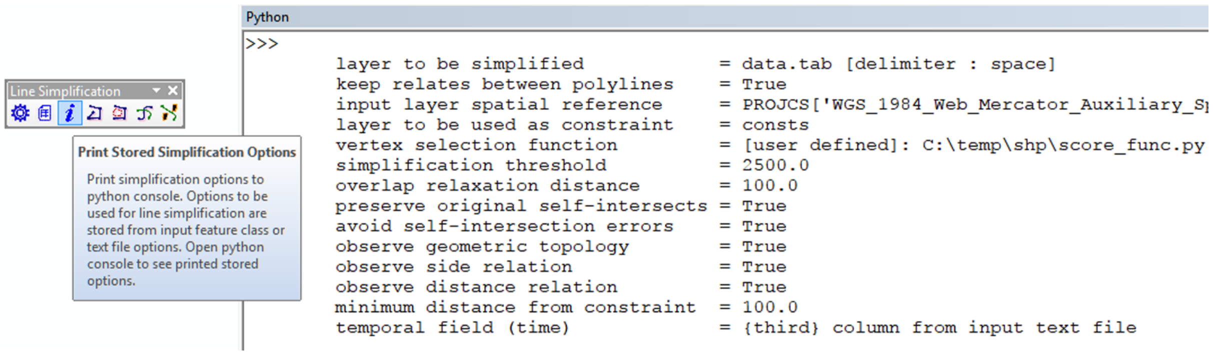
Figure 5.9: Add-In TextFile Options

### 5.1.5 Simplification Options

Options for constrained and unconstrained simplification are itemized in Algorithm 0. From the Add-In toolbar, it is possible to print stored simplification options using *print stored simplification options* tool. To use the tool, open the *Python* window in ArcMap to see options in console (see Figure 5.10).

**Algorithm 5.4** Constrained DP Options

```
$threshold \leftarrow$ float
$mindist \leftarrow$ float
$relaxdist \leftarrow$ float
$keep\_selfintersects \leftarrow$ boolean
$avoid\_new\_selfintersects \leftarrow$ boolean
$geom\_relation \leftarrow$ boolean
$dir\_relation \leftarrow$ boolean
$dist\_relation \leftarrow$ boolean
```


Python
Line Simplification
Print Stored Simplification Options
Print simplification options to python console. Options to be used for line simplification are stored from input feature class or text file options. Open python console to see printed stored options.

```
>>>
        layer to be simplified              = data.tab [delimiter : space]
        keep relates between polylines      = True
        input layer spatial reference       = PROJCS['WGS_1984_Web_Mercator_Auxiliary_S
        layer to be used as constraint      = consts
        vertex selection function           = [user defined]: C:\temp\shp\score_func.py
        simplification threshold            = 2500.0
        overlap relaxation distance         = 100.0
        preserve original self-intersects   = True
        avoid self-intersection errors      = True
        observe geometric topology          = True
        observe side relation               = True
        observe distance relation           = True
        minimum distance from constraint    = 100.0
        temporal field (time)               = {third} column from input text file
```

Figure 5.10: Add-In Print Options

#### 5.1.5.1 Custom Vertex Selection Function

The difference between DP and SED algorithm is how the *score* function calculates the offset of a vertex from the generalized line (start and end point). The DP algorithm computes minimum distance between a generalized line and intermediate vertices, whereas SED computes the distance between an intermediate vertex and its spatio-temporal trace on the generalized line. Also in Fig. 5.11, we show a sample script as a DP maximum offset function. Figure 5.11 can be used as a sample for user defined vertex selection function; see Figures 5.8 and 5.9.

```python
"""
score with signature : score(coords, rng) :returns (int, float)
"""
import  arcpy
srs = arcpy.SpatialReference(3857)

def Pt(o):
    return arcpy.PointGeometry(arcpy.Point(*o[:2]), srs)

def segment(a, b):
    pta, ptb = Pt(a), Pt(b)
    if pta.equals(ptb): #Note:if a == b polyline((a, b), srs) is empty
        g = pta
    else:
        array = arcpy.Array([arcpy.Point(*o) for o in (a[:2], b[:2])])
        g = arcpy.Polyline(array, srs)
    return g

def score(pln, rng):
    """
     Finds maximum  offset vertex from the
     general segment(pln[i] -- pln[j]) where
     i, j is rng[0] and rng[1]
     `:param: pln` coordinates of polyline
     `:param: rng` index of vertices, slice of pln from (i, j)
     `:returns: (int, float)`
    """
    seg = segment(pln[0], pln[-1])
    index, offset = rng[1], 0.0

    if rng[1] - rng[0] > 1:
        for k in xrange(rng[0] + 1, rng[1]):
            dist = seg.distanceTo(Pt(pln[k]))
            if dist >= offset:
                index, offset = k, dist

    return index, offset
```

Figure 5.11: Custom Score Script

## 5.2 Case Study

Contours present a topological challenge during simplification. Contour lines are not allowed to self-intersect or intersect with neighbours. This means the simplification of a contour requires observing the simplification of other neighbouring contours, the simplified geometries must preserve the spatial relationship of the original polylines. Using a snapshot of USGS contours near Aspen, Colorado, we show the difference between constrained simplification at 40m (avoid self-intersection, keep relationship between input polylines). See Figures 5.12, 5.13 and 5.14.

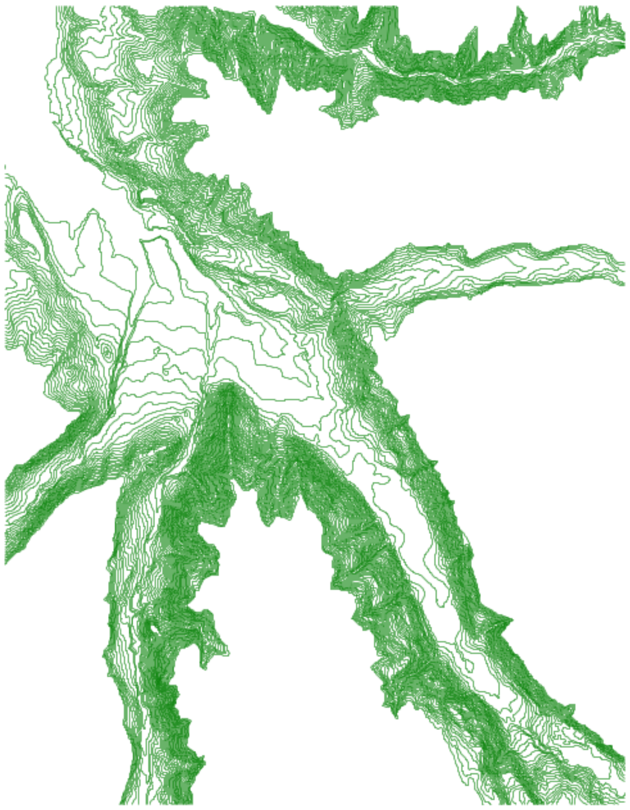


Figure 5.12: Original contours at 40$m$ interval

The original (Fig. 5.12) snapshot of contours contain 3,242,100 vertices. The table bellow show comparable space saving in constrained and unconstrained simplification at thresholds of 20, 30 and 40 meters. Despite comparable space savings, the constrained version preserves the original spatial relationship between contours whereas the unconstrained may result in self-intersection or intersections with other contours at higher thresholds (see Fig. 5.13 and 5.14).

| Vertices at Threshold | $20m$ | $30m$ | $40m$ |
|---|---|---|---|
| Constrained | 14249 (99.56%) | 11393 (99.65%) | 9911 (99.69%) |
| Unconstrained | 14193 (99.56%) | 10887 (99.66%) | 8935 (99.72%) |

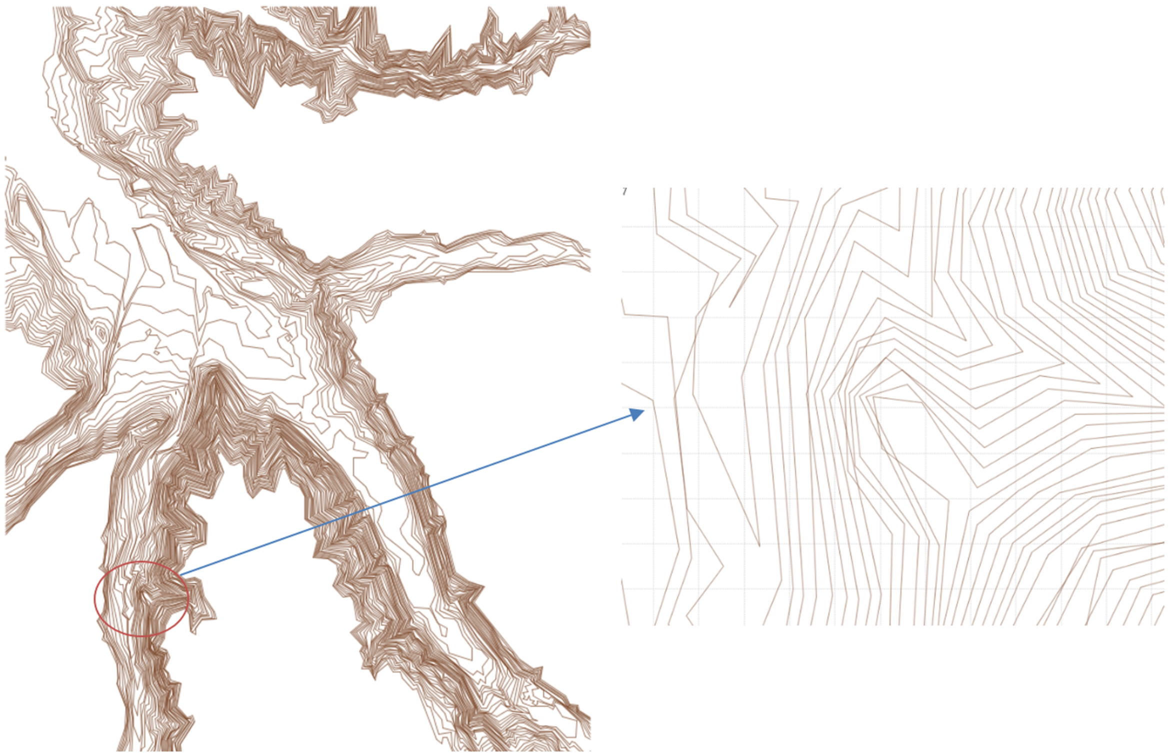


Figure 5.13: Unconstrained DP simplification at $40m$

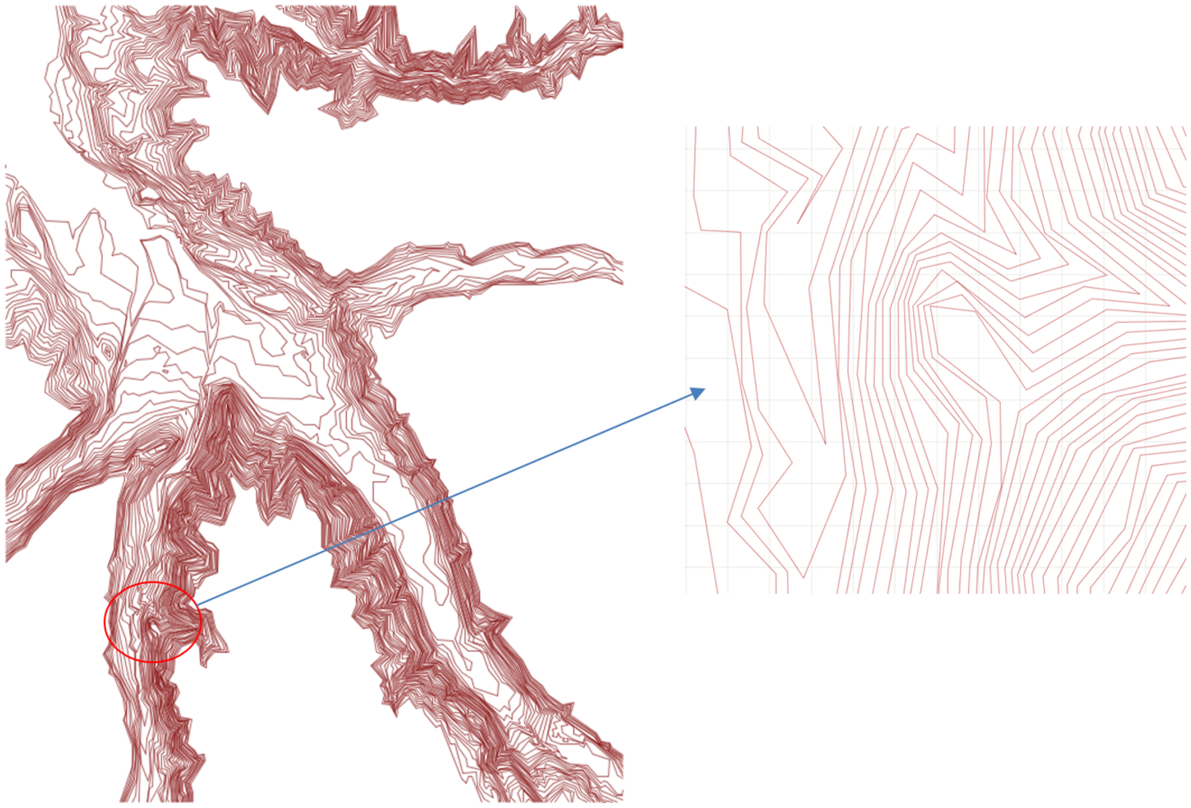


Figure 5.14: Constrained DP simplification at $40m$

## 5.3 Summary

In this application, we implemented a constrained line simplification algorithm for linear features using ArcGIS ArcPy. The constrained implementation utilized the Douglas-Peucker (DP) algorithm as a general framework for polyline decomposition. The implementation was extended to support a variant of the DP algorithm for spatio-temporal polylines: the Synchronized Euclidean Distance (SED) [Douglas and Peucker, 1973; Meratnia and Rolf, 2004; Tienaah et al., 2018]. To make this research practical, we have implemented a python Add-In to extend ArcGIS line generalization toolset. Features of the Add-In include:

1. simplification of spatially dependent and independent linear features,
2. user defined offset plug-in for different variants of the DP algorithm,
3. prevention of self-intersection as result of simplification,
4. preservation of planar self-intersection,
5. a fine grained control on displacement of non-planar intersection, and
6. preserves intersect/disjoint, homotopy, and distance relations to other geometries as constraints.

Users of desktop ArcGIS can use this tool to simplify arbitrary static and dynamic (spatiotemporal) polylines with other layers as contextual planar constraints.

# References

Douglas, D. H. and Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122.

Meratnia, N. and Rolf, A. (2004). Spatiotemporal compression techniques for moving point objects. In *International Conference on Extending Database Technology*, pages 765–782. Springer.

Tienaah, T., Stefanakis, E., and Coleman, D. (2018). Line simplification while keeping it simple or complex. In *Line Simplification Under Spatial Constraints*. Geographic Information Systems and Science - Research Lab, Geodesy and Geomatics Engineering, UNB.

# Chapter 6:

# Summary and Conclusion

Line simplification is a well studied research problem in cartography, geographic information systems, computer graphics, and computational geometry. Despite its long history, it is still an unsolved problem. In fact, some authors Guibas et al. [1993]; Estkowski [1998]; Estkowski and Mitchell [2001] have proofed the hardness of the simplification problem in the context maintaining topological consistency for simple polylines. Based on its intractable nature, the purpose of this research is the development of spatial data structures and geometric heuristics to provide a practical solution under multiple spatial and topological constraints. Our primary focus is simplification of arbitrary polylines in the context of arbitrary planar objects independent of error function as way of extending our implementation to spatiotemporal simplification.

## 6.1 Research Summary

Chapter 1 starts this research with an introduction, related work, research problem, scope, and objectives. The chapter presents a literature review and lays out the structure of this dissertation.

In Chapter 2, we develop a constrained Douglas-Peucker algorithm using a polyline to be simplified and other geometries as contextual constraints. The focus was to develop a planar contextual model that incrementally “untangle" using relevant characteristic vertices to resolve topological conflicts. The implementation demonstrated a consistent technique to accelerate multi-scale simplification of polylines in context of other geometries.

Chapter 3 develops and implements a novel set of geometric heuristics for the $Min{-}\#$ line simplification problem with contextual planar objects (point, polylines, polygons) as constraints. This chapter improves on the work done in Chapter 2 by avoiding unnecessary introduction of original vertices during topological conflict resolution. Given an arbitrary polyline, a set of planar objects, and $\varepsilon > 0$, we implement a consistent topological simplification by observing the following constraints:

1. preserve planar and non-planar intersections,
2. avoid introducing new self-intersections as a result of simplification,
3. preserve intersect/disjoint relations to planar objects, and
4. preserve homotopy in simplification.

our experimental evaluation showed a competitive compression ratio compared to unconstrained simplification.

In Chapter 4, we study contextual online simplification of arbitrary trajectory streams by extending algorithms and heuristics in Chapter 3 to the spatiotemporal domain. In our online streaming setting, we have a limited amount temporal memory, and assume an infinite external storage. The ordered sequence of possibly an infinite incoming spatiotemporal instants of a moving object forms its trajectory. Our online implementation is constrained to avoid introducing new self-intersections as a result of simplification. It also preserves self-intersection between simplification units. Other constraints include disjoint, intersect, proximity, and homotopy in the context of arbitrary planar objects. Experimental evaluation was conducted using real world data - moving vessels in the Aegean Sea with islands as planar constraints.

Our results demonstrate a higher compression savings with topological errors in unconstrained simplification. Experimental results showed a compression difference of approximately 4% between constrained and its corresponding unconstrained simplification. This shows a competitive constrained compression ratio in less than four times the processing time of unconstrained simplification.

Chapter 5 implements heuristics and algorithms developed in this research in a commercial GIS package. The purpose of this implementation is to extend the functionality of ESRI ArcGIS cartographic toolset. This contribution to desktop GIS provides a fine grained simplification of polylines and spatiotemporal trajectories using another layer as contextual constraints.

## 6.2 Research Contributions

Out-of-context (unconstrained) simplification is fast with a higher compression ratio but leads to topological, proximity, and other spatial relational errors. A consistent constrained simplification provides the benefits of data reduction while preserving the original static/spatiotemporal characteristics in the input polyline. The contributions of this research dissertation to line simplification under spatial and topological constraints are realised through data structures, algorithms, and heuristics that:

1. prevent self-intersection in simple polylines,
2. preserve planar self-intersection (vertices with degree $> 2$) in complex polylines and between groups of polylines,
3. preserve non-planar self-intersection at some distance offset ($\delta$) between simplification units of one or more polylines,
4. avoid topological linear inversion during simplification,

5. preserve homotopy relation between a polyline and its simplification in the context arbitrary planar objects,
6. preserve intersect or disjoint relation between a polyline and other planar objects,
7. maintain a minimum distance relation to arbitrary planar objects, and
8. offer online topological (self-intersection and homotopy) simplification of trajectory streams with support for ad hoc queries.

Finally, using principles and techniques developed through this research, a cartographic tool to support contextual simplification for static and dynamic polylines was created to extend ESRI ArcGIS cartographic toolbox.

## 6.3 Limitations and Recommendations

Algorithms and data structures in Chapters 2 and 3 are implemented in-memory, and this is not scalable for datasets that will not fit in main memory (RAM). Future work should explore I/O efficient structures and algorithms for constrained polyline simplification.

In this dissertation, we explored the $Min{-}\#$ problem. Given $\varepsilon > 0$, the $Min{-}\#$ problem computes $L'$ that uses the smallest number of vertices among all $\varepsilon$-approximations of $L$. There is limited research on contextual topological simplification based on the $Min{-}\varepsilon$ problem: find $L'$ with at most $K$ vertices that minimizes $\varepsilon$ over all approximations of $L$ that have $K$ vertices.

Our external implementation in Chapter 4 can be improved by using a external key-value storage versus an object relational database (PostGRE/PostGIS). Furthermore, an efficient implementation using a systems programming language (without the overhead of memory management - garbage collection) will improve running time and resource utilization. Future work should explore hard real-time spatiotemporal simplification for moving object trajectories with topological constraints.

## 6.4 Conclusion

In this dissertation, we consider the $Min-\#$ simplification problem. We developed a novel set of geometric heuristics to demonstrate how to avoid self-intersection introduced by a simplification algorithm (RDP, SED), preserve planar and non-planar intersection, constrain contextual intersect/disjoint relation, observe proximity constraint, and homotopic simplification of arbitrary polylines in the context of arbitrary planar objects.

The algorithms and heuristics are evaluated using synthetic and real world datasets for static and spatiotemporal polylines. The experimental results show a fast convergence to a constrained simplification with competitive compression ratio as compared to unconstrained requirements. Our contributions (Section 6.2) and practical implementations provides an approach to handle multiple topological constraints during automated line simplification. Our heuristics provide an insight into contextual simplification to support cartographic generalization, graphic visualization, and other data reduction domains that use static and spatiotemporal polylines. Source code and resources of this dissertation are made available at *github.com/TopoSimplify*.

# References

Estkowski, R. (1998). No steiner point subdivision simplification is np-complete. *In Proceedings of the 10th Canadian Conference on Computational Geometry*.

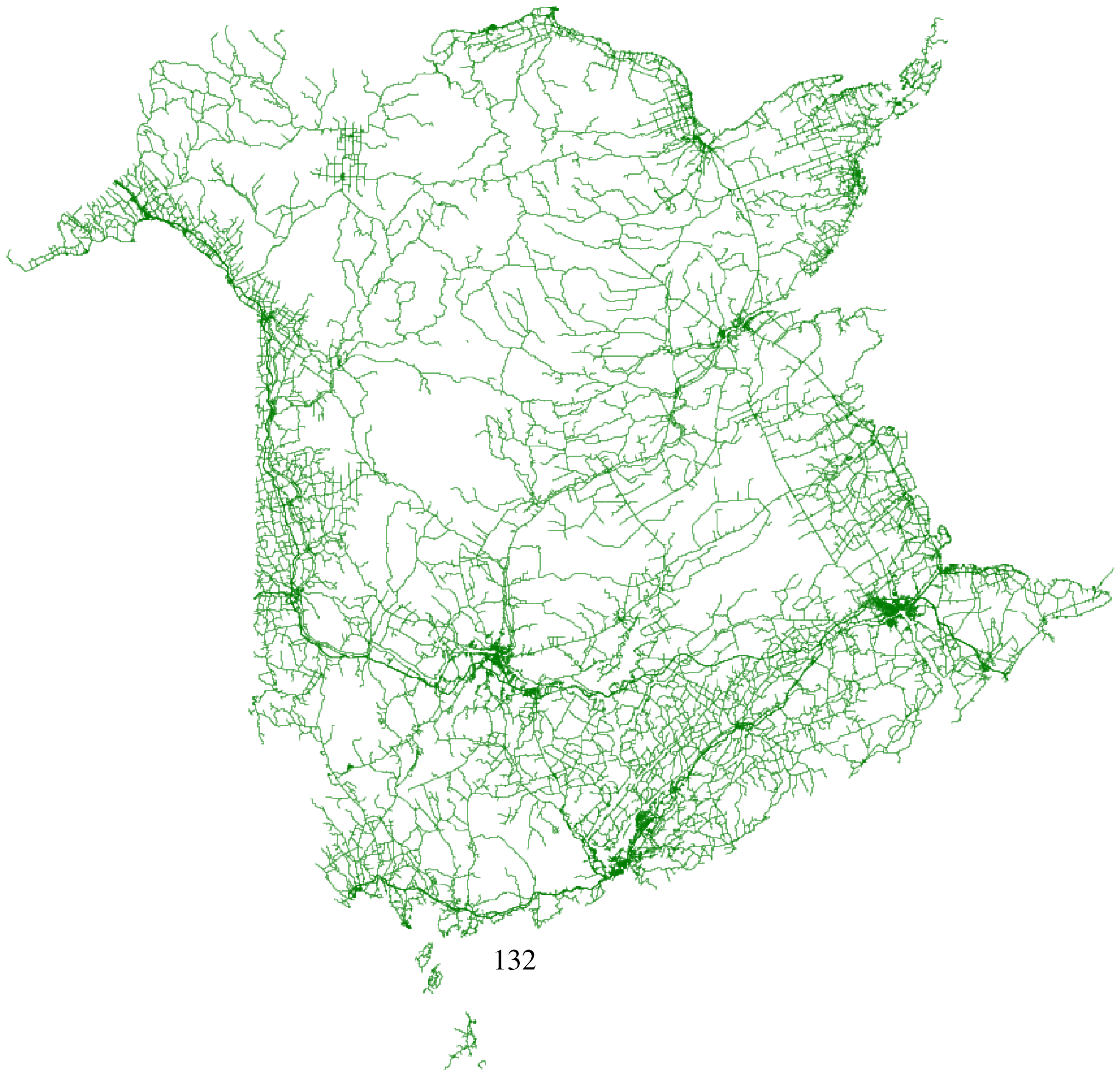
Estkowski, R. and Mitchell, J. S. (2001). Simplifying a polygonal subdivision while keeping it simple. In *Proceedings of the seventeenth annual symposium on Computational geometry*, pages 40–49. ACM.

Guibas, L. J., Hershberger, J. E., Mitchell, J. S., and Snoeyink, J. S. (1993). Approximating polygons and subdivisions with minimum-link paths. *International Journal of Computational Geometry & Applications*, 3(04):383–415.

# Appendix A:

# Appendix A

## A.1 New Brunswick Road-Network



# Appendix B:

# Appendix B

## B.1 Pitkin County - Colorado, US

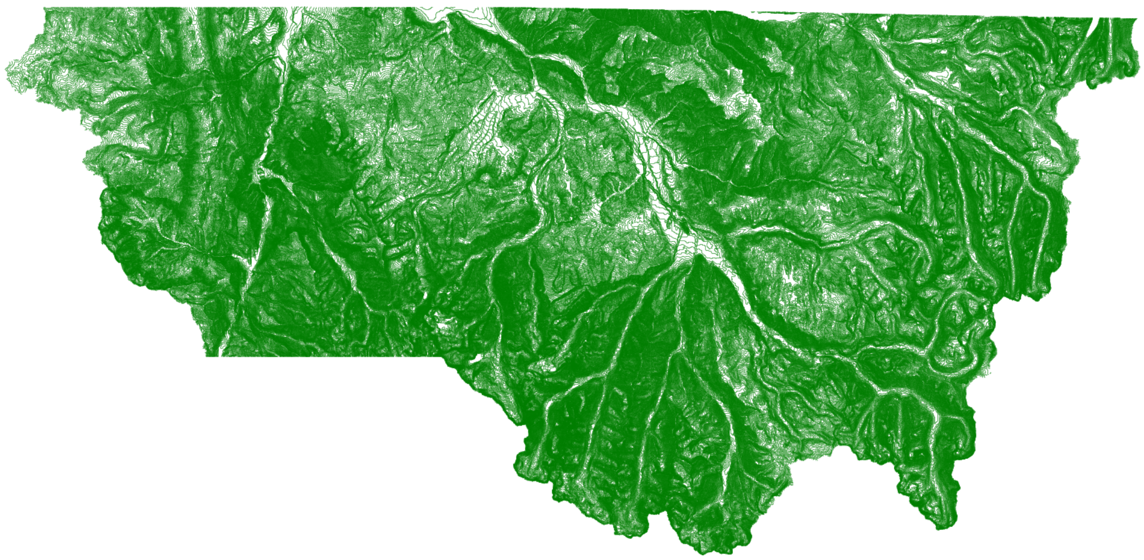


Figure B.1: Pitkin County $40m$ Contours

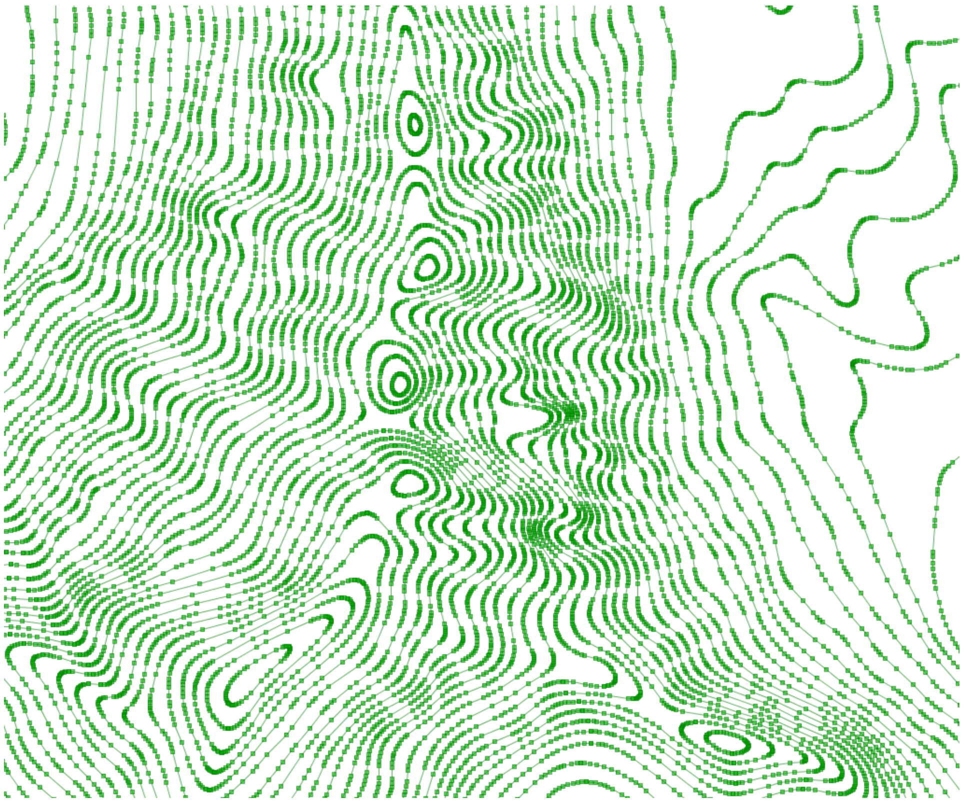


Figure B.2: Pitkin County $40m$ Contours - A snapshot of vertices

# Vita

**Candidate's full name**: Titus Tienaah

**University attended**:

- *2011-2018* - **PhD** Candidate, Geodesy and Geomatics Engineering, University of New Brunswick, Fredericton, Canada.

- *2008-2011* - **MScE**, Geodesy and Geomatics Engineering, University of New Brunswick, Fredericton, Canada.

- *2003-2007* - **BSc**, Geomatics Engineering, Kwame Nkrumah University of Science and Technology, Kumasi, Ghana.

**Publications**:

1. Tienaah, T., Stefanakis, E., & Coleman, D.(2018). Line Simplification While Keeping it Simple or Complex ... *In Review*
2. Tienaah, T., Stefanakis, E., & Coleman, D.(2018). Topologically Consistent Online Trajectory Simplification ... *In Review*
3. Tienaah, T., Stefanakis, E., & Coleman, D.(2015). Contextual Douglas-Peucker Simplification, Geomatica 69(3)327-338.

**Conference Presentations & Proceedings**:

1. Tienaah, T., Stefanakis, E., & Coleman, D. (2015). Contextual Line Generalization-Extending ArcGIS Generalization Toolset. In Proceedings of the 18th AGILE international conference on geographical information science (pp. 9-12).

2. Tienaah, T., and Stefanakis, E., (2014). Troy is ours - How on earth could Clytaemnestra know so fast?. In the Proceedings of the 17th AGILE Conference on Geographic Information Science, Castellon, Spain.

3. Tienaah, T.(2014). Real-time Linear Simplification under Space Constraints. In Proceedings of Spatial Knowledge and Information, Banff, Canada.

4. Sutherland, M. & Tienaah, T. & Seeram, A. & Ramlal, B. & Nichols, S.(2013). Chapter 7: Public Participatory GIS, Spatial Data Infrastructure, and Citizen-Inclusive Collaborative Governance. Global Spatial Data Infrastructure Association Press, pp. 123-140.

**Research Projects**:

1. Tienaah, T., and Stefanakis, E., (2015-2017). Constrained Line Simplification(CLS) for ArcGIS. Engage Project, Industrial partner: Esri Canada.

2. Tienaah, T., Rak, A. and Coleman, D. (2013). An Examination and Critical Comparison of Alternative Maintenance Models for the Nova Scotia Digital Topographic Database. Contract Report of 2-year consulting study undertaken for the GeoNova Program Office, Service Nova Scotia and Municipal Relations, Province of Nova Scotia. March.

3. Mioc, D., Anton, F., Nickerson, B., Santos, M., Adda, P., Tienaah, T., et.al. (2011). Flood progression modelling and impact analysis. In Efficient Decision Support Systems-Practice and Challenges in Multidisciplinary Domains. InTech.