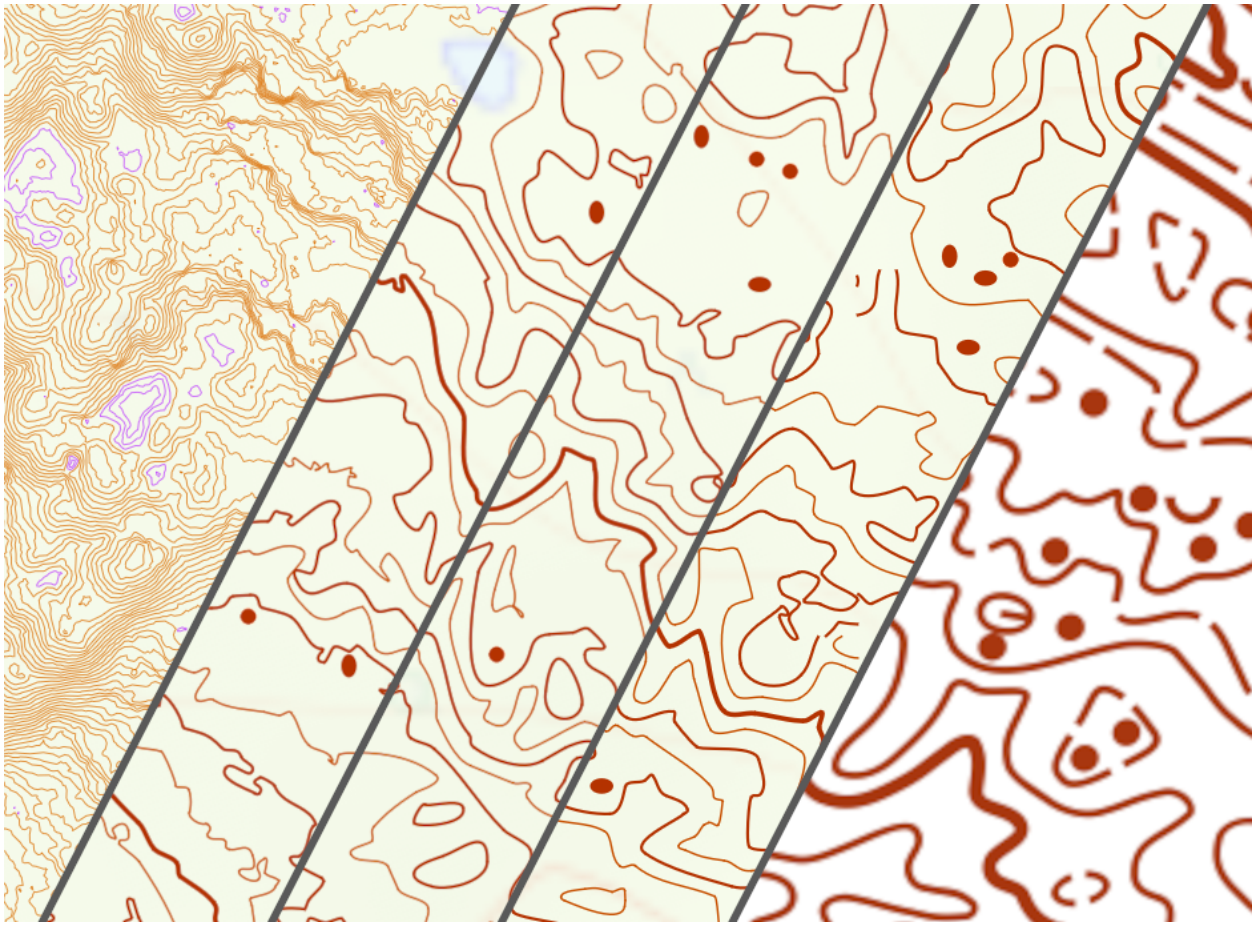


Contour extraction from digital elevation models for topographical mapping with minimal information loss

Master's thesis in Complex Adaptive Systems

Øyvind Hjermstad

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's Thesis in Complex Adaptive Systems

Contour extraction from digital elevation models for topographical mapping with minimal information loss

Øyvind Hjermstad

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
Gothenburg, Sweden 2025

Contour extraction from digital elevation models for topographical mapping with minimal information loss
Øyvind Hjermstad

Supervisor: Marco L. Della Vedova, Department of Mechanics and Maritime Sciences
Examiner: Marco L. Della Vedova, Department of Mechanics and Maritime Sciences

Master's Thesis 2025
Department of Mechanics and Maritime Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Cover: Different contour extraction methods applied to a detailed hillside in Norway. The methods are: basemap, raw contours, smoothed contours, naive interpolation correction method, hand-made contours

Typeset in LaTeX
Gothenburg, Sweden 2025

Contour extraction from digital elevation models for topographical mapping with minimal information loss
Øyvind Hjermstad
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology

Abstract

This thesis defines a measure of the amount of information lost when extracting contours from a Digital Elevation Model (DEM). The measure is based on triangulation of the contour vertices and Sibson's natural neighbor interpolant $Z^{(1)}$. The length of the gradients at the triangulation points are derived from the natural neighbors defined by the triangulation itself, clamped by a minimal length. The gradient directions are, by definition, perpendicular to the contours. As contours are used to visualize a surface, it is important that the plot is not overly cluttered and impedes readability. This constraint is added to the optimization problem by defining a measure of the squiggliness of a contour. The measure is inspired by the bending force needed to deform a thin metal rod to the shape of the contour. Different methods are applied to the task of minimizing the information loss and contour squiggliness and their performances are compared.

An application is created for DEM generation from Lidar data, contour extraction and visual comparison of the extracted contours. The application aids in parameter tuning for the tunable models. The extracted contours are also visually compared to hand-made contours by professional orienteering mappers.

Sammanfattning

Denna uppsatsen definerar ett mått på mängden information som förloras när höjdkurvor extraheras från en digital höjdmodell (DEM). Måttet baseras på triangulering av höjdkurvornas hörnpunkter vid hjälp av Sibson's naturliga granninterpolant $Z^{(1)}$. Längden på gradienterna längs kurvorna deriveras från trianguleringen och begränsas nedåt av en minimilängd. Gradientriktningarna är, per definition, normal mot höjdkurvorna. Eftersom höjdkurvor används för att visualisera en yta är det viktigt att representationen inte blir överdetaljerad och därmed försämrar läsbarheten. Detta restriktionen läggs till optimeringsproblemet genom att definiera ett mått på en hur krokig en höjdkurva är ("squiggliness"). Måttet på krokigheten är inspirerat av den böjkraft som krävs för att deformera en tunn metallstav till kurvans form. Olika metoder tillämpas för uppgiften att minimera informationsförlusten och kurvernas krokighet, och deras resultat jämförs.

En applikation har utvecklats för att generera av DEM:arna från Lidar-data, extrahering av höjdkurvor samt visuell jämförelse av de extraherade kurvorna. Applika-

tionen underlättar parameterjustering för modellerna med justerbara parametrar. De extraherade höjdkurvorna jämförs också visuellt med handritade kurvor gjort av professionella orienteringskartritare.

Sammendrag

Denne masteroppgaven definerer et mål på informasjonen som går tapt når høydekurver ekstraheres fra en høydemodell (DEM). Målet er basert på interpolering av hjørnepunktene til de ekstraherte høydekurvene ved hjelp av Sibsons naturlige nabointerpolant $Z^{(1)}$. Lengden på gradientene deriveres fra trianguleringen selv, men er begrenset av en minimumslengde. Gradientenes retning er, per definisjon, normal mot høydekurvene. Ettersom kurver brukes for å visualisere en overflate er det viktig at kurvekartet ikke blir overfyllt av kurver som dermed reduserer lesbarheten. Denne restriksjonen blir lagt til i optimeringsproblemet gjennom å definere et mål på hvor krokete en kurve er ("squiggliness"). Målet er inspirert av kraften som skal til for å bøye en tynn metallstav til formen av høydekurven. Olike metoder blir benyttet for å minimere informasjonstapet mens kurvene forblir lite krokete og resultatene sammenlignes.

Et program har blitt utviklet for hele prosessen fra å generere DEMene fra Lidar-data til å ekstrahere høydekurvene og for å sammenligne de ulike metodene. Programmet gjør det enkelt å justere parametere for de modellene som er justerbare. De ekstraherte høydekurvene blir også visuellt sammenlignet med håndlagde høydekuver laget av professjonelle orienteringskarttegnere.

Keywords: Contour line, isoline, topographical map, orienteering mapping

Preface

This report presents the result of my master's thesis project carried out at the Department of Mechanics and Maritime Sciences at Chalmers University of Technology during the winter of 2024/25.

Acknowledgements

I would like to thank my supervisor and examiner Marco L. Della Vedova.

Øyvind Hjermstad, Gothenburg, April 2025

List of Acronyms

Below is the list of acronyms that are used throughout this thesis, listed alphabetically:

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
COPC	Cloud Optimized Point Cloud
CRS	Coordinate Reference System
DEM	Digital Elevation Model
EPSG	European Petroleum Survey Group
EPT	Entwine Point Tile
GIS	Geographic Information System
GUI	Graphical User Interface
LiDAR	Light Detection And Ranging
RDP	Ramer-Douglas-Peucker
ReLU	Rectified Linear Unit
TIN	Triangulated Irregular Network
TPI	Topographic Position Index
WKT	Well Known Text

Nomenclature

Below is the nomenclature of sets, parameters, and variables that have been used throughout this thesis.

Wording

Isolines	mathematical contours
Contour lines	generalized contours
Contour set	the union of contour lines of different isolevels

Sets

$\mathcal{C}$	Contour set
$\mathcal{P}$	Set of points

Parameters

λ	Weighting coefficient

Variables

$\boldsymbol{u}$	Vectors are given in bold lowercase,
$\boldsymbol{H}$	Matrices are given in bold uppercase,
$\boldsymbol{\nabla}$	Gradient vector,
$\boldsymbol{u}^x$	The x-component of a vector,
$\hat{\boldsymbol{u}}$	A normalized vector,
P_i	Point i in a point set,
$\boldsymbol{u}_i$	the 2D-coordinates of point P_i,
z_i	the value of point P_i,

Contents

List of Figures

List of Tables

1
Introduction

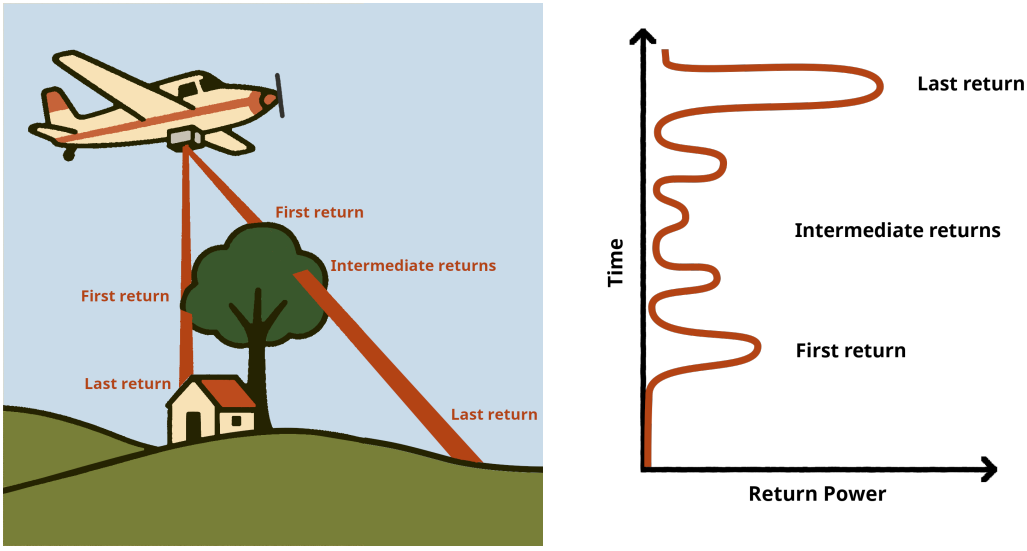
1.1 Background

A level set is the subset of a real-valued function's domain where the function takes a constant value, called an isolevel.

$$L_c(f) = \{\boldsymbol{x} \mid f(\boldsymbol{x}) = c\} \tag{1.1}$$

In two dimensions, level sets are curves called contour lines or isolines. The isolines of continuous functions are continuous curves or points. Contour lines are a fundamental tool for visualizing real-valued functions of two variables in two dimensions. A union of contour lines of different isolevels provides an intuitive visualization of the underlying scalar field and will in this thesis be called a contour set.
Contour sets are used in many fields, such as meteorology, geology, hydrology, physics, and topography. In the context of topography contour lines are curves of equal elevation. Topographic maps are important data sources with many applications, including military, environmental science, urban planning, civil engineering, sport, recreation and help and rescue missions. They enable a clear visualization of the landforms, and can be used to derive elevation, steepness, aspects, accessibility and potential dangerous areas.
Most topographic maps uses a fixed contour interval where the contours are allowed to deviate from the true isolines defined by (1.1). The only way for isolines to achieve a better description of the area between any two contours is by decreasing the contour interval, which quickly leads to cluttered and unreadable maps. By allowing the contour lines to deviate from their true positions, can the readability and explaining power of the contours as a set be increased, even though the accuracy of single contours by themselves decrease. By taking into account how experienced map users interpret contour lines can more information about the landforms in between neighboring contours can be embedded into the contours.

The most accurate elevation data collection method, from which contours can be extracted, is LiDAR [1]. LiDAR (or, simply, "lidar") is an acronym for light detection and ranging and is an active remote sensing technology in which the return time and intensity of an emitted laser pulse are used to infer information such as the position of the reflecting surface. Aerial lidar, where the lidar unit is carried by a plane, drone or helicopter, is the most common method for collecting lidar data over large areas [2]. The data resulting from an aerial lidar scan is a point cloud with millions of points returned from the ground, vegetation, buildings, cars and



(a) A lidar unit carried by a plane scans over the terrain and collects the returned laser pulses which are processed into points.

(b) Example power distribution of the returned signal from the right pulse in subfigure a.

Figure 1.1: A simplified diagram of the lidar scanning process. Multiple lidar points can be derived from a single emitted laser pulse by analyzing the returned laser power distribution over time.

other objects. The point cloud is often filtered for noise and unwanted objects such as cars, animals, humans and noise points. The remaining points can be classified based on their characteristics. The most common classification filtering is ground classification where points on the bare earth are marked with the ground class label. More lidar classes exist, such as buildings, water and vegetation, but only ground points are of interest for contours. The ground classified points can be used to describe the bare ground elevation in the interior of the point cloud either through triangulation into a triangulated irregular network (TIN) or by gridding to a digital elevation model (DEM). Both TINs and DEMs facilitate easy extraction of contours, but they have different strengths and weaknesses. A DEM is a raster representation of the bare earth elevations. Each pixel of the raster holds an elevation value. Some information is bound to get lost in the creation of a DEM. However, a TIN preserves all the information in the ground returns of the lidar point cloud. This is not really a problem for our purpose as the point density of lidar data is high enough that the extracted DEM will contain enough information. The regular structure of DEMs make them easier to work with and makes processing faster and is the chosen data format for elevations in this thesis.

Various automated methods to increase the readability of a contour set exist. These methods can broadly be divided into two groups: contour-editing and DEM-editing. Hybrid approaches do also exist and those will be considered as a part of the second group. However, no method has successfully replicated the amount of information about the landform that human mappers are able to embed in a set of contours.

The central challenge in generating useful topographic contours lies in balancing the need for simplification, to efficiently being able to convey information, while preserving the essential topographic information of the landform. To address this challenge, an understanding of how experienced map readers interpret contours needs to be embedded into the contouring algorithm.

1.1.1 Interpreting contours

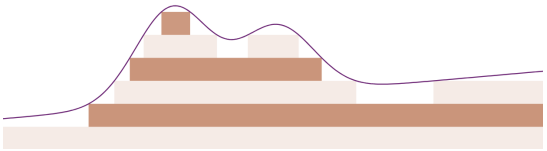
The four main rules for interpreting contour maps (with a constant contour interval) are listed below.

- Contours are curves of constant level, the gradient is therefore perpendicular to the contour.
- Bunched up contours implies larger gradients and spread out contours implies smaller gradients.
- Hachures (or slope lines) are tick marks on a contour indicating the downhill direction and are used to clarify downhill from uphill where it is not directly obvious.
- Closed contours should be interpreted as knolls, i.e. the inside is above the surroundings, unless hachures are used to mark a depression.

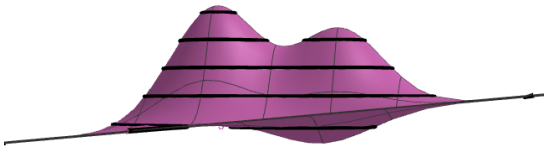
It might seem logical that every contour will eventually close in on it self, but this is in general not true. However, every contour will on a closed domain (such as a sphere) eventually close in on it self, assuming a continuous underlying function. This means that all contours on the earth form closed curves as elevation changes are continuous (ignoring cliff overhangs) and the earth is topologically equivalent to a sphere.Maps usually only consider a small part of the Earth, or in most cases, a flat projection of a small part of the Earth, and contours on maps may therefore not be closed.
An illustrative representation of contours is depicted in Figure 1.2, where the function f defined in equation (1.2) is used as a model of the terrain.

$$f(x,y) = 4.2e^{-(x^2+y^2)} + 3e^{-((x-2)^2+y^2)} - e^{-((x-2)^2+(y+2)^2)} + 0.2x \tag{1.2}$$

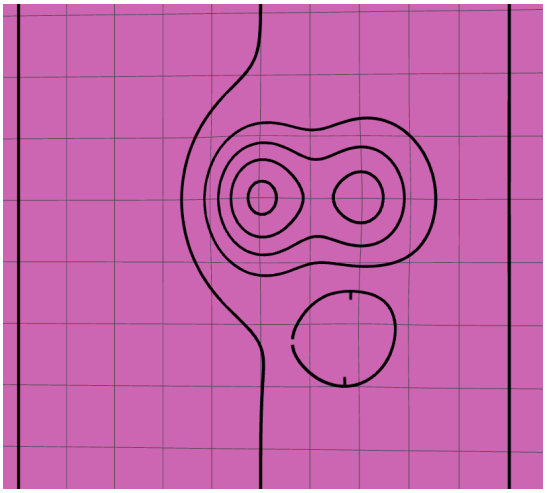
Figure 1.2a shows a cross section of f, specifically $z = f(x, 0)$. The function is shown in purple and the contours are shown as stacked blocks. The function is steep where the blocks' ends are stacked tightly and less steep where they are spread out. This stepped interpretation of the contours is neither similar to the actual function nor to any landforms in general (except maybe the rice terraces of southeast Asia). A more natural way to read the contours would interpolate the values in between the contour lines based on the steepness.
Extending this into two dimensions we get the side profile of $z = f(x, y)$ in figure 1.2b. A contour representation of f, $L_\kappa = \bigcup_{c\in\kappa}\{(x, y) \mid f(x, y) = c\}$, $\kappa = \{-1, 0, 1, 2, 3, 4\}$, is shown in figure 1.2c. Notice how the closed contour in the bottom of the image makes use of hachures to mark it as a depression. An experienced map reader would, from the contours in figure 1.2c, quickly be able to visualize f in similar way to the perspective view in 1.2d.



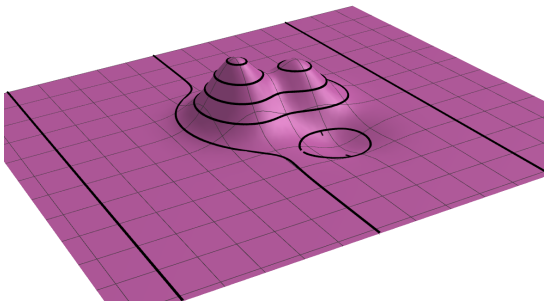
(a) A cross sectional view of $f(x, 0)$. The contours are visualized as steps.



(b) f in side profile.



(c) A top down view of f. The same as a contour visualization of f.



(d) A perspective view of f.

Figure 1.2: Different views of f given by (1.2) with black contours with interval 1 unit.

1.2 Purpose

The purpose of this thesis is to investigate the possibility of using interpolation error as a measure of the quality of a contour set and to use that measure to improve on contour sets. Contours are the most basic and information rich element on many topographic maps. Improving the automatic generation of contours has the potential of increasing the amount of information carried by maps around the world. The second purpose is to simplify the start of the map-making process by creating an application for generating basemaps from lidar data.

1.3 Goals

The goal of this thesis is two-fold. The first goal is to define a measure of the information loss in contour extraction Digital Elevation Model (DEM) while the generated contours are nicely behaved and investigate methods of optimizing this loss metric.
The second goal is to produce an application that will facilitate visual comparison

of different contouring algorithms. This application should take lidar point clouds as input, extract a DEM from the point data and generate contours from the DEM. The application should have a graphical user interface for easy and quick iteration on algorithm parameters. The application should be built in such a way that extending it to become a general purpose automatic orienteering map generation tool should be straightforward. The application should be able to generate topographic maps from the decided on parameters. The application should write georeferenced vector map files oriented to the magnetic north in the XML-based omap format of OpenOrienteering Mapper[1]. The omap format was chosen because it is the native format of the free open-source OpenOrienteering Mapper software. This makes the written map files directly accessible to everyone in a professional and specialized mapping application. Omaps XML-structure also makes constructing a file writer straightforward.

1.4 Limitations

Form lines are intermediate contours used in topographic maps that are placed at the halfway point between contour lines to better describe important local terrain features that the regular contour interval is too coarse to be able to represent. Form lines should be used sparingly so as not to clutter the map and can be discontinued where they are not needed. Choosing where to use form lines is itself a difficult problem, and excessive use can easily worsen the readability of the map. This thesis will not take form lines into account, and no form line algorithm will be implemented. However, a possible algorithm for form line selection will be proposed in appendix A.1. The algorithm is based on the same objective measure of the goodness of a contour set that this thesis aims to define, but will not be implemented due to time constraints.

Hachures (also called slope lines) are tick marks on the lower side of a contour used to differentiate between depressions and knolls on topographic maps. Closed contours on topographic maps are always assumed to be knolls unless hachures are used. Hachures might also be added to complex landforms to make it immediately obvious to the map users what is up and what is down. While knowing when hachures are needed for depressions is as simple as checking the winding order of the contour, is placing the hachure more difficult. Hachures should be placed so they are easy to spot and increase the clarity of the interpreted elevation image while not cluttering the map and reducing readability. The international specification for orienteering maps[2] (ISOM) recommends hachures to be placed in re-entrants when possible. Deciding where hachures should be placed outside of depressions is a hard task. The placement of hachures will not be considered in this thesis.

[1] See `https://www.openorienteering.org/apps/mapper/`

[2] See `https://orienteering.sport/iof/mapping/`

1.5 Organization of the content

The remainder of the thesis is organized as follows: Chapter 2 presents earlier research and the theory on which this work is based. Chapter 3 presents the method for defining an objective function for scoring contour set. The following chapter, chapter 4, investigates and presents methods for using the objective function as an optimization target. The chapter on results, chapter 5, compares the performance of the objective function with subjective contour scores and compares the results of the different optimization methods. Appendix A presents a possible algorithm for using the objective scoring function for creating form lines, the post-processing of the contours and details for writing georeferenced omap files.

2
Theory

This chapter investigates previous research on the topic of contour generalization while preserving information and lays the theoretical foundation upon which the next chapters build. The theory covers sections cover DEM interpolation from lidar and contours, contour extraction and simplification, and the fundamentals of relevant machine learning techniques.

2.1 Previous Research

While there is not much research done on optimizing a contour set with respect to an interpolation-based error measure, there is research on various methods to create contours that are suited for visualization by eliminating noise and features too small for the scale. These methods can broadly be divided into two groups; methods that modifies the DEM before contour extraction and methods that work on the raw contours directly.

2.1.1 DEM modification

Modifying the DEM prior to contour extraction is a common approach for generating smoother and more generalized contour lines suitable for visualization [3] [4] [5] [6] [7]. The fundamental idea is to apply filtering techniques to the grid data, effectively removing high-frequency noise and small-scale topographic variations. These methods operate directly on the elevation values, implicitly influencing the resulting contour geometry. Basic approaches often involve simple low-pass filtering, adapted from image processing. For instance, a mean filter replaces each grid cell's elevation with an unweighted average of its neighbors within a defined window. Mean filters offer computational efficiency, but tend to isotropically blur sharp topographic features like ridges and valleys. A Gaussian filter computes a weighted average, where the weights decrease with distance according to a Gaussian function, resulting in smoother transitions and better control via the standard deviation parameter. The filter remains isotropic and can still excessively smooth significant features if applied aggressively. A major drawback of low-pass filters is their indiscriminate smoothing, which can remove or distort significant features. To address this, adaptive smoothing techniques adjust the filtering based on local terrain characteristics.

Topographic Position Index (TPI) weighted smoothing quantifies a point's elevation relative to the average elevation of it's neighbors and identifies features like

ridges or valleys, allowing for less smoothing in these critical areas and more in planar regions [3]. It is not without fault though, as TPI based smoothing might smooth excessively in areas such as saddles.
Another adaptive approach is local entropy smoothing [4]. The local information entropy is used as a measure of the terrain complexity and guides the filter to apply less smoothing in complex, high-entropy areas and more in simpler, low-entropy zones.
Line integral convolution is a technique originally developed for visualizing vector fields by placing densely packed lines following the direction of the field, but has later been adapted to terrain generalization [7]. The method successfully removes details and retains sharp mountain ridges and preserve sharp transitions to at areas while offering control over the level of detail in mountain slopes.
Another approach using line energy to smooth a DEM is based on minimizing the curvature of polylines fitted to the DEM grid [5]. The method uses hard accuracy bounds to limit the amount a single pixels value can change.

Normal field smoothing

Normal field smoothing is a form of adaptive smoothing that considers surface normals derived from the DEM [6]. This approach smooths the surface normal field across neighboring normals that are pointing in a similar direction and applies elevation update steps in multiple iterations to the DEM to make it conform with the aligned normals. This has an anisotropical smoothing effect. The DEM is smoothed more strongly in directions of small curvature and less across sharp edges to preserve the definition of crest lines and valley bottoms.
Normal field smoothing is inspired by 3D mesh denoising approaches applied to TINs by Sun et al. [8]. Normal field smoothing takes the idea of using the geometric information of surface normals in the triangulation for smoothing and applies it to the inherent structure of raster grids, which has computational advantages over working triangulations.
The algorithm computes the surface normal vector field from the input DEM. For each grid cell, the partial derivatives $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$ are calculated from the elevation values in the 3-by-3 neighborhood of the cell. The surface normals can be obtained from the partial derivatives.
This initial normal field undergoes a smoothing operation implemented via spatial convolution filtering using a specified kernel size. The smoothing operation is feature preserving because of the algorithms weighting mechanism. When calculating the smoothed normal for a central cell, the contribution of each neighboring cell within the kernel is weighted based on the angular difference between their normals. A user-specified threshold angle, θ_t, acts as a discriminant: neighbors whose normal vector deviates from the center cell's normal by an angle greater than or equal to θ_t are assigned zero weight, effectively isolating the smoothing process from cells across significant terrain breaks. For neighbors within the threshold are weights assigned proportionally to the similarity of their normals, ensuring that smooth transitions are averaged while discontinuities are respected.
The algorithm iteratively updates the DEM's elevation values to conform to the smoothed normal field. In each iteration, a new elevation for a target cell is com-

puted as a weighted average derived from the implicit tangent planes defined by the smoothed normals of its neighboring cells. Critically, this averaging process re-employs the same feature-preserving weighting scheme based on the angular difference between the smoothed normals and the threshold θ_t. Neighbors with a normal angle difference greater than θ_t are deemed to be across a feature boundary and do not contribute to the elevation update of the target cell. This iterative refinement continues for a user-specified number of iterations, progressively aligning the elevation surface with the smoothed geometric representation captured by the normal field. The resulting output is a DEM where fine-scale roughness is attenuated, while significant features like channel banks, terrace scarps, and gully edges are maintained.

2.1.2 Contour modification

An alternative or complementary strategy involves processing the contour lines directly after their extraction from the DEM. These methods operate on the vector representation (sequences of vertices) using geometric algorithms for simplification, smoothing, and feature removal. Low-pass filtering concepts can be applied to the sequence of contour vertices, treating it as a two-dimensional signal. One method transforms the vertex coordinates into the frequency domain using the Fourier Transform, allowing high-frequency components associated with noise and small wiggles to be attenuated or removed before transforming back to the spatial domain; however, sharp cut-offs can sometimes introduce artifacts like the Gibbs phenomenon [9]. A simpler and widely used technique is sliding window averaging, where each vertex position is adjusted based on the average position of its neighbors along the contour within a defined window. This local averaging smooths small variations effectively, but can cause contour shrinkage in regions of high curvature. Related geometric simplification algorithms, such as the Ramer-Douglas-Peucker algorithm, primarily aim to reduce vertex count but inherently perform a degree of smoothing by removing points contributing to small deviations, see section 2.3.2.1. Another powerful approach utilizes splines to generate smooth, aesthetically pleasing curves that approximate the original contour vertices, replacing piecewise linear segments with mathematically defined curves. Bézier splines are common in computer graphics and fit piecewise polynomial segments to the vertices [10]. Splines are often favored for their local control property and smoothness. Snakes, or Active Contour Models, originating from image analysis, employ energy-minimizing splines that iteratively evolve to balance internal energy (promoting smoothness, penalizing curvature) and external energy (attracting the curve to the original data points) [11]. While adaptable, snakes can be computationally intensive and sensitive to initialization. Axial Splines combine the energy minimization of Snakes method with generalized axial symmetry and has been shown to significantly improve noisy contours derived from lidar data [12]. Spline-based methods generally produce high-quality, smooth contours, though the choice of method involves trade-offs regarding computational cost, control, and the desired geometric properties of the final curves.

2.1.3 Karttapullautin

Karttapullautin (KP) is a command line application created by Jarkko Ryyppö for generating orienteering maps from ground classified lidar data [13]. KP has seen great success in generating highly detailed maps of entire countries, such as Estonia, Finland, Norway, Spain and Sweden, with France and New Zealand having partial coverage[1]. The program generates contours, cliffs, detects buildings and maps vegetation densities. We will only focus on the contouring algorithm of KP.
KPs contouring algorithm is a collection of algorithms that work together performing modifications on both the DEM level and the extracted contours. The most interesting part of the algorithm is the knoll detection. During the knoll detection KP checks the DEM for distinct knolls, i.e. knolls with a steepness and a relief over certain thresholds, and marks them with a contour even though they might not be situated at a isolevel. This is an example of letting the contours deviate from the isolines to represent the landform better than the isolines are able to.
While KP is the state of the art in contour generation, is it still far off the quality of hand drawn maps. The main issue is that the contours are too constrained to the isolines and are not able to deviate from the isoline to capture the landform outside of the knoll detection.

2.2 Raster and vector data

Two primary data models for an effective representation of real-world geographic features are used within Geographic Information Systems (GIS). These are called raster and vector data models. These models encode geographic location and spatial relationships in different ways and have complementary strengths and weaknesses.
The vector data model describes geographic features through the use of geometric objects, such as points, polylines, and polygons. These geometric primitives are defined by a set of coordinates defined in a coordinate reference system and can be tagged with extra data in the form of attributes.
A point signifies a discrete location using a single coordinate and is on topographic maps used for representing features such as boulders and prominent individual trees.
Polylines are used to depict linear features. They are defined by an ordered sequence of vertices connected by straight segments and may or may not form a closed loop. Examples include representations of contours, roads, paths and small rivers.
Polygons are two-dimensional objects that delineate features with defined boundaries that enclose an area. A polygon consists of a closed polyline that forms the outer boundary and any number of closed polylines that form holes in the polygon. These are commonly used to represent features such as lakes, distinct vegetation zones, agricultural areas or private areas.
Vector data allows for high positional precision, limited primarily by the accuracy of the original data capture, and when used for representing discrete features are vector data very memory efficient. However, the vector model is inherently less suitable for representing continuously varying phenomena, such as elevation or temperature.

[1] Visit mapant.net for links to the country-covering maps

The raster data model approaches the geographic space as a continuous field, representing it through a regular grid of cells or pixels that completely cover a defined geographic area. Every cell within this grid has a uniform size, which determines the model's spatial resolution, and holds a numeric value that quantifies the value of the phenomenon at that location. The raster is normally defined by the coordinates of the top left corner in some coordinate system, the size of a single cell and the number of rows and columns in the grid, as well as the raster values. Cell values are often floating-point numbers, ideal for representing continuously varying phenomena such as elevation values in a DEM.
The raster model offers distinct advantages. Its structural simplicity, essentially a matrix, makes it straightforward to understand and process. It excels at representing continuous surfaces and fields, making it the natural choice for data such as elevation or imagery.
Many powerful analytical operations are computationally efficient due to the regular grid structure. The raster model, like the vector model, is not without limitations. Its spatial precision is inherently constrained by the cell size or resolution. All variation within a single cell is ignored, potentially obscuring features smaller than the cell resolution. Raster datasets, particularly those covering large areas at high resolution, can require significant storage space, although various compression techniques can be employed to limit this issue.
The optimal choice of data model depends on the specific nature of the geographic phenomenon. Discrete, clearly demarcated entities like property boundaries, buildings, or road networks lend themselves well to the vector representation. Conversely, continuous or field-like phenomena such as elevation, temperature distributions or remotely sensed imagery are more appropriately modeled using the raster approach.
Non-imagery rasters are usually assigned a color ramp and each cell a value from the ramp based on the cell value. The visualization will saturate if the span of the raster values is far greater than the number of discernible colors in the ramp. In saturated rasters will discerning small features from the surroundings be hard or impossible depending on the total range of the raster values. Note that the number of discernible colors in a continuous color ramp also is highly individual and thus not very well suited for maps. Furthermore, topographic maps cannot use multiple raster based data models as it would quickly become illegible trying to combine multiple continuous fields, such as the elevation field and vegetation density, on the same map. Conversion between raster and vector formats – rasterization (vector-to-raster) and vectorization (raster-to-vector) – may therefore be needed.
Vectorizing the raster into contours that can efficiently be used to visualize the data is common. However, this process is not without information loss.
This process is not unique to elevation data, but as that is the focus of this thesis will only elevation data and DEM rasters be considered in the continuation.

2.3 Contour extraction

Extracting contours from a DEM represented as a raster grid of elevation values, involves identifying paths along which the elevation value remains constant. This process transforms the discrete grid representation into a vector representation of

the terrain's shape.

2.3.1 Marching Squares

The Marching Squares algorithm is a common method for generating contour lines from 2D scalar fields, such as a DEM grid [14]. It operates by considering cells of four adjacent grid points (2x2 squares) at a time.
For each 2x2 square, the algorithm determines which vertices are above or below the desired isovalue. Based on the configuration of these vertices (there are $2^4 = 16$ possible configurations), the algorithm determines how the directed contour line segments pass through that square. Each configuration corresponds to a predefined way of drawing line segments connecting the edges of the square. The corner values of the edges are linearly interpolated to find the precise point on the edge where the contour line crosses.

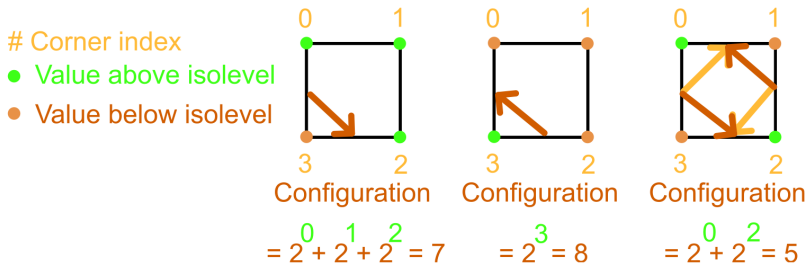


Figure 2.1: The contour extraction process for a single cell in 2x2 square in the DEM. Based on the configuration of what corners are above or below the isolevel can an index be calculated and the type of contour segment to be added is uniquely defined by that index.

From the last configuration in figure 2.1 two line segments should be created, but two different sets of line segments can be chosen. Both the brown and the yellow line segments are valid contours for configuration 5 and one set must be chosen. Different techniques can be employed to deal with this issue. One solution is to check which diagonal has vertices the furthest from the isolevel and choose the line segments perpendicular to that diagonal. A simpler approach is to always choose one set. The advantage of the first approach is having greatest distance between the two line segments, while the second approach simplifies the implementation without much difference in the output.
This problem also for occurs for configuration 10. Notably, all configurations are symmetric with respect to 15 i.e. configurations k and $15-k$ are identical, but with the line segments reversed.
The algorithm "marches" through the grid row by row, processing each 2x2 square and connecting the line segments generated within each square to form continuous contour lines.

Marching Squares is a fundamental algorithm for extracting isolines directly from raster data. However, as the corner values of the evaluated squares are linearly

interpolated, it assumes that the raster data models a piecewise linear function. This can lead to clipping of the function in areas of high curvature leading to some information loss. This can be minimized by increasing the raster resolution. The resulting contours can also be overly detailed and jagged, often requiring simplification.

The origins of the Marching Squares algorithm is not known, but the first examples dates back to the early 1980s. Marching Squares' three dimensional equivalent is called Marching Cubes. Marching Cubes was introduced by Lorensen and Cline in 1987 and is among the most cited papers in computer graphics ever with a wide range of applications including visualizations of medical imaging [15].

2.3.2 Simplification

Contours generated directly from algorithms like Marching Squares can contain an excessively large number of vertices, making them computationally intensive to store, process, and render. Furthermore, the finest details on the contours might represent noise or artifacts from the DEM rather than significant terrain features. Simplification algorithms aim to reduce the number of vertices in a polyline while preserving its essential shape within a specified tolerance.

2.3.2.1 Ramer-Douglas-Peucker

The Ramer-Douglas-Peucker (RDP) algorithm is a widely used method for polyline simplification. The algorithm was introduced independently by Ramer in 1972 [16] and Douglas and Peucker in 1973 [17]. It is one of the earliest successful algorithms developed for cartographic generalization [18]. While it is not the most performant polyline simplification algorithm, it is the best in terms of simplification while minimizing line displacement [18].

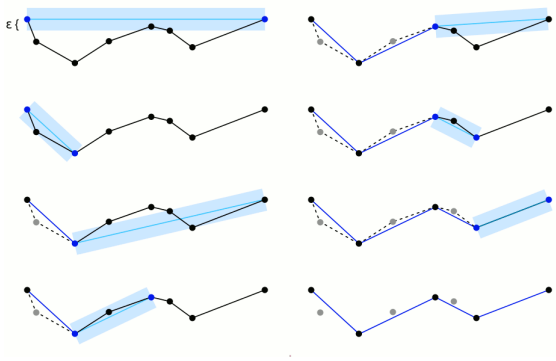


Figure 2.2: The Ramer-Douglas-Peucker algorithm. The polyline is recursively split at the vertex furthest from the blue line. If all points are inside the buffer then only the end points are kept. Adapted from Douglas-Peucker Animated.gif by Mysid from Wikimedia Commons. CC BY-SA 3.0

The algorithm starts with a line segment connecting the start and end points of the polyline. It then finds the intermediate vertex that is farthest from this line

segment. If the vertex is further from the straight line than a specified tolerance, ϵ, then the vertex is kept and the algorithm recursively applies itself to the two polyline segments starting or ending at this vertex. If the maximum distance is within the tolerance, all intermediate points are discarded, and the segment is represented just by its endpoints.
RDP is effective at reducing the data volume while maintaining the general shape of the line. However, the choice of ϵ is important. A too large ϵ can lead to significant information loss, potentially smoothing over important small features or altering the perceived shape of the line. RDP is known for preserving critical points, but can sometimes create visually sharp angles where the initial line was changing smoothly.

2.3.2.2 Bezier curves

Bezier curves are parametric curves defined by a set of control points. While not a direct simplification algorithm in the same way as RDP, are splines of Bezier curves used for smoothing polylines and for efficient representation of smooth shapes.
Bezier curves were in the late 50's and early 60's independently developed and applied to graphical design by Paul de Castelju and Pierre Bezier who used them for designing the body work of Citroen and Renault cars, respectively. The mathematical basis for the curves was developed in 1912 by the Russian mathematician Sergei Bernstein [10].
Bezier curves are defined as a linear combination of the Bernstein polynomials (2.1) of a chosen degree n [19]:

$$b_{v,n}(t) = \binom{n}{v} t^v (1-t)^{n-v}, \quad \text{for } v = 0, \ldots, n \quad \text{and } t \in [0,1] \tag{2.1}$$

The most used curves in computer aided design are cubic Beziers:

$$B_3(t) = \sum_{v=0}^{3} P_v b_{v,3}(t) = P_0(1-t)^3 + P_1 3t(1-t)^2 + P_2 3t^2(1-t) + P_3 t^3 \tag{2.2}$$

The coefficients P_i are the control points of the Bezier. Setting t to 0 and 1 it can be seen that P_0 and P_n are the start and end points of the curve. A Bezier curve of degree n is defined by $n+1$ control points. The intermediate P_i are called the handles of the curve and defines the shape of the curve without the constraint that the curve must pass through them.
For polyline smoothing or simplification is a series of Bezier segments fitted to the original polyline, this is called a Bezier spline. When fitting Bezier splines to polylines are P_0 and P_n commonly fixed at the start and end vertices of the current segment. For cubic Beziers, the handles P_1 and P_2 are located on the tangents of the polyline at P_0 and P_3, respectively. The handle positions along the tangents are found by solving a least-squares problem with respect to the distance between the polyline vertices and the Bezier. If the squared error is too large, the polyline is split at the vertex of maximum error and the Bezier fitting is applied on the two parts recursively.
Cubic Bezier splines produce smooth curves that represent polylines efficiently given the assumption that the polyline models a smooth phenomena such as contour lines.

The fit of a Cubic Bezier spline to a polyline is dependent on the choice for the maximum allowed error. Smaller allowed error leads to more Bezier segments that follow the polyline closer, but at the expense of a larger number of points in the line.

2.4 Voronoi Diagram and Delaunay triangulation

A Voronoi diagram is a partition of a space $E \subseteq \mathcal{R}^m$ that given a set of n points, P_i, tiles the space such that each tile T_k is the subset of E with P_k as the closest point [20], see figure 2.3a. The Voronoi tile of point P_i is given by equation (2.3):

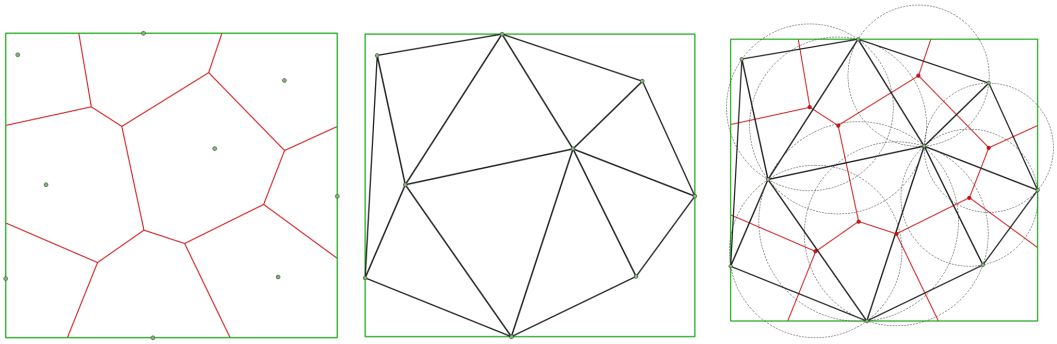
$$V(P_i) = \{\boldsymbol{x} \mid \|\boldsymbol{x_i} - \boldsymbol{x}\| < \|\boldsymbol{x_j} - \boldsymbol{x}\| \; \forall \; j \neq i\} \tag{2.3}$$

Each condition $\|\boldsymbol{x_i} - \boldsymbol{x}\| < \|\boldsymbol{x_j} - \boldsymbol{x}\|$ describes an open half-hyperplane bounded by the perpendicular bisector of P_i and P_j. The intersection of all $n-1$ bisectors from the $n-1$ conditions describes a tile T_i which will be an open convex polyhedra. The union of the tiles cover the entirety of E expect at the tiles' boundaries. Data points who's Voronoi tiles touch are called neighbors. This definition of a neighbor is completely data determined and the set of neighbors of a point defined in this way will be called the natural neighbors of that point. Each data point will in m dimensions have at least $m+1$ neighbors, as long as the tile does not meet the boundary of E.

The Delaunay triangulation is constructed by drawing a line from all of the points to their neighbors, see figure 2.3b. The convex hull of the points will always be a subgraph of the Delaunay triangulation as the tiles of points sharing an edge on the convex hull will always touch, just not necessarily inside E. This can be seen from the fact that the tile boundaries are perpendicular bisectors of the Delaunay edges, see figure 2.3c.
The resulting triangulation has some favorable characteristics. A circumhypersphere defined by the points of any simplex in the triangulation is guaranteed to not contain any other points of the triangulation [21]. If we only considered triangulations in the plane, which is what will be applied in this work, it can be proven that the Delaunay triangulation is the triangulation which maximizes the minimal angle in any triangulation of the input points. This makes the triangulation suitable for interpolation, as long and skinny triangles are avoided where possible. Long and skinny triangles would be bad for interpolation as that would make far apart points natural neighbors of each other and therefore violate the principal of proximity upon which many interpolation methods build. The Delaunay triangulation forms the basis for many interpolation methods.

2.5 Interpolation

Interpolation is the process of estimating a function value at an unsampled location in the interior of the sampled locations based on known values at sampled locations.



(a) The Voronoi tiles of a point set on $E \subset \mathbb{R}^2$.

(b) The corresponding Delaunay triangulation of the points.

(c) Both the Voronoi tiles and Delaunay triangulation with circumcirles.

Figure 2.3: The Voronoi tiles and Delaunay triangulation of a point set on $E \subset \mathbb{R}^2$. The Delaunay triangulation is the dual graph of the Voronoi tiles. Notice how the centers of the circumcircles are on the intersections of three Voronoi edges and that the Voronoi edges are perpendicular bisector of the Delaunay edges.

Interpolation is fundamental both in the creation of DEMs from the scattered lidar points and in recreating a DEM from a contour set. Almost all interpolation methods build on the heuristic that close data points should be more similar than far apart data points and can be broadly classified based on whether they use all available data points or only a subset of nearby data points. Some interpolation methods only work on data points structured regularly, such as on a grid, while others work on scattered data regardless of the structure. One of the typical use cases of scattered data interpolants is gridding the data as gridded data often is easier to process. This section will only consider scattered data interpolants as both lidar and contour data are scattered and not regular.
Global interpolation methods use all available data points in the dataset to estimate the value at a new location. The advantage of global support methods is that they utilize the maximum amount of information available, but a significant drawback is that they can be computationally intractable for large datasets.
Local interpolation methods, on the other hand, estimate the value at a new location using only a subset of available data points. They assume that much of the information stored in far away points is either irrelevant or also contained in near-by points. Local interpolants are more computationally efficient than global methods, but limits the amount of information used in interpolation.

2.5.1 Inverse distance weighting

Inverse distance weighting (IDW) is a simple interpolation method that estimates function values as a weighted average of values at known points. IDW is an exact interpolator, meaning that the interpolated surface passes through all the known data points and as such does not account for noise in the input data points. The weight assigned to each known point is inversely proportional to the distance to the

interpolation location, raised to a power p [22]. That is $w_i = 1/\|\boldsymbol{x}_i - \boldsymbol{x}\|^p$, for some distance metric $\|\cdot\|$ and power p. A higher power p gives more influence to closer points. IDW can be implemented with either global or local support. The IDW interpolant is defined in equation (2.4).

$$Z^{\text{IDW}}(\boldsymbol{x}) = \frac{\sum_i w_i z_i}{\sum_i w_i}, \quad \text{with } w_i = \frac{1}{\|\boldsymbol{x}_i - \boldsymbol{x}\|^p} \tag{2.4}$$

IDW is an intuitive interpolant and easy to implement. However, it has limitations. The choice of the power parameter p, distance metric and the neighborhood definition significantly affects the result. Due to the mathematics of weighted averages, can IDW not estimate values outside the range of the sample data. The quality also degrades with uneven point distributions.

2.5.2 Kriging

Kriging is a global geostatistical interpolation method that utilizes the statistical relationship between points based on distance and direction (spatial autocorrelation) of the data to determine the optimal weights for interpolation.
Kriging involves two main steps. The first step is to model the spatial structure of the data using a variogram, which describes how data similarity decreases with distance. The second step is to use the variogram model to calculate the weights for a linear combination of the known values to predict the value at a new location. Kriging aims to provide the best linear unbiased predictor (BLUP), minimizing the estimation variance [23].
Kriging is considered more advanced and statistically robust than IDW because it incorporates spatial correlation. It provides not only predictions but also estimates of prediction uncertainty, modeling the input values along with gaussian noise. Different types of kriging exist to handle different assumptions about the data's mean or structure, where the more general assumptions demand heavier computations. Universal kriging is the most general and must be considered to be comparable to the interpolants later on. Kriging is particularly useful when data exhibits clear spatial trends or anisotropy (directional dependencies). While the resulting interpolated values are superior to IDW, are they much more computationally intensive to compute and requires careful modeling of the variogram. However the need for large matrix inversions makes this approach intractable for lidar data.

2.5.3 Natural neighbour interpolation

Natural neighbour interpolation are a group of local interpolation methods based on the natural neighbors defined by the Delaunay triangulation.
To make the neighbor relation quantitative, Sibson introduced a coordinate set he called local coordinates [24]. Consider a Voronoi diagram generated by the points $\mathcal{P}$ on E. For a tile T_m generated by the point P_m, subdivide the tile into subtiles T_{nm} where T_{nm} is the subset of T_m with P_n as the second closest point. The T_{nm} are, for the same reasons as T_m, open, convex and polyhedral. T_{nm} is non-empty only if P_n and P_m are neighbors. Let $\kappa(n)$ denote the volume of T_n and $\kappa_m(n)$ the volume of

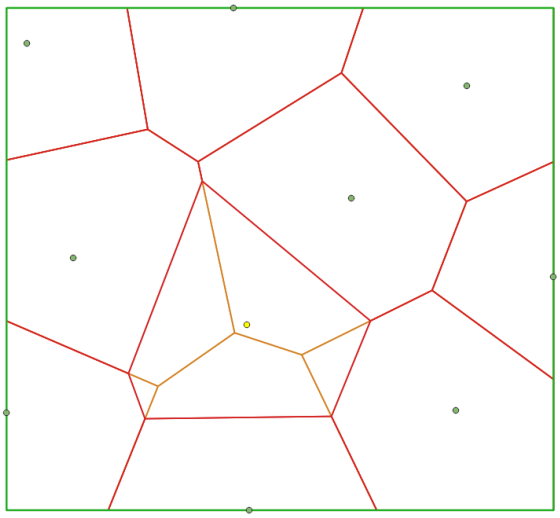


Figure 2.4: The yellow point P has been inserted into the triangulation for interpolation. The red diagram is the new Voronoi tiles and the orange are the boundaries of the subtiles of the T_P. The boundaries coincide with the old Voronoi boundaries before P was inserted.

T_{nm}. $\lambda_m(n) = \kappa_m(n)/\kappa(n)$ is the fraction of T_m with P_n as the second closest point and are a measure of the neighborliness of P_m as a neighbor to P_n.
The neighborliness of P_m and P_n are zero together, but need not be equal if positive. This defines a quantitative measure for the neighborliness of any two data points in the triangulation, but we are also interested in the measure for any point not in the triangulation. This can be achieved by inserting the interpolation point into the triangulation and continuing as above.

Lets call the interpolation point P at position $\boldsymbol{u}$. The neighborliness of P and P_m is then $\lambda_m(\boldsymbol{u}) = \kappa_m(\boldsymbol{u})/\kappa(\boldsymbol{u})$, where $\kappa(\boldsymbol{u})$ is the volume of the Voronoi tile of P. The $\lambda_i(\boldsymbol{u})$ are called the local coordinates of $\boldsymbol{u}$ and have some favorable properties. $\lambda_i(\boldsymbol{u})$ is a continuously differentiable function of $\boldsymbol{u}$ except at the data points $\boldsymbol{u} = \boldsymbol{u}_i$. The coordinates also fulfill an identity called the barycentric coordinate property (BCP), which states $\sum_m \lambda_m(\boldsymbol{u})\boldsymbol{u}_m = \boldsymbol{u}$ as long as the tile of P, inserted at $\boldsymbol{u}$, does not meet the boundary of E. The BCP states that the $\lambda_m(\boldsymbol{u})$ express $\boldsymbol{u}$ as a convex combination of its neighbors [25]. This opens the door for using the local coordinates as a basis for interpolation.
It should be mentioned that other choices of local coordinates exists; such as the Laplace coordinates, which uses the boundary volumes of the tiles instead of the face volume to define its coordinates, or barycentric coordinates, which express any point on the inside of a triangle as a linear combination of its vertices. These coordinates are not considered in this work as they produce interpolants of lower quality.

2.5.3.1 Sibson $Z^{(0)}$

From Sibson's local coordinates can a simple weighted averaging interpolant be constructed (2.5) [24]. The weights are based directly on the proportionate areas

stolen in the Voronoi diagram by inserting the interpolation point.

$$Z^{(0)}(\boldsymbol{u}) = \sum_i \lambda_i(\boldsymbol{u}) z_i \tag{2.5}$$

where $\lambda_i(\boldsymbol{u})$ are defined as proportion of the Voronoi tile belonging to P_i that would belong to P if P was to be inserted into the Delaunay triangulation at $\boldsymbol{u}$ and z_i the function value at data site P_i.
The $Z^{(0)}$ interpolant is C^1 continuous everywhere except at the data points, where it is only C^0. This interpolant reduces to piecewise linear interpolation in one dimension and it can reproduce any linear function in any number of dimensions correctly. The region of influence of a data point P_i is the volume for which an inserted point would acquire P_i as a neighbor. This region is defined as the union of all the circumspheres of the geometric simplexes in the triangulation which P_i are a part of. In two dimensions are the circumspheres circles and the simplexes triangles 2.5.

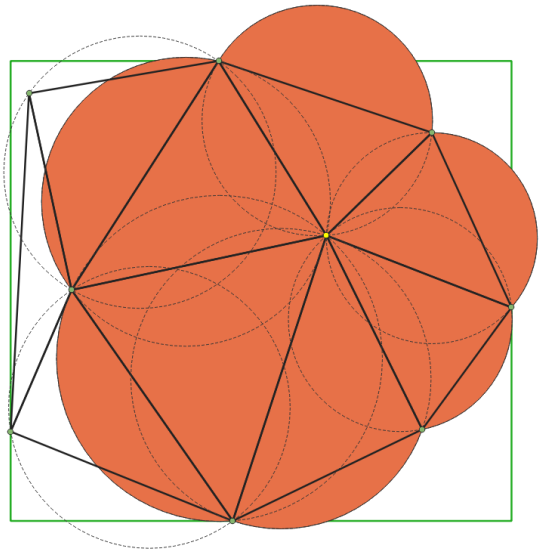


Figure 2.5: The region if influence for the yellow point. Notice that even though the region of influence covers areas outside of the triangulation, is the interpolant not defined outside of the convex hull of the data points.

As this is a weighted averaging interpolant is the interpolated value bound to the range of the data points. Of course this is not a desired feature of the interpolant and to get around that gradients at the data points are needed.

2.5.3.2 Sibson's Gradient Estimation

Many scattered data sources do not have gradients defined at the data points. However, gradients can be approximated from the values of the natural neighbors. Sibson proposes a method to approximate the gradient, $\boldsymbol{\nabla}_i$, of P_i by fitting the first degree function through $(\boldsymbol{u_i}, z_i)$ that minimize the squared distance to the natural

neighbors of P_i by solving a least squares problem [24].

$$\boldsymbol{\nabla}_n = \boldsymbol{H}_n^{-1}\boldsymbol{p}_n, \tag{2.6}$$

$$\boldsymbol{H}_n = \sum_m \lambda_m(n)\frac{(\boldsymbol{u}_m - \boldsymbol{u}_n)(\boldsymbol{u}_m - \boldsymbol{u}_n)^T}{(\boldsymbol{u}_m - \boldsymbol{u}_n)^T(\boldsymbol{u}_m - \boldsymbol{u}_n)}, \tag{2.7}$$

$$\boldsymbol{p}_n = \sum_m \lambda_m(n)\frac{(\boldsymbol{u}_m - \boldsymbol{u}_n)(z_m - z_n)}{(\boldsymbol{u}_m - \boldsymbol{u}_n)^T(\boldsymbol{u}_m - \boldsymbol{u}_n)}, \tag{2.8}$$

With gradients at the data points can stronger interpolants be constructed.

2.5.3.3 Sibson $Z^{(1)}$

While Sibson's $Z^{(0)}$ is C^1 except at data points can we with addition of gradients at the data points construct an interpolant which is C^1 everywhere. Sibson calls this interpolant $Z^{(1)}$ [24]. It is based on the idea that the interpolant can be approximated by a linear function $\zeta(\boldsymbol{u})$ in the vicinity of each data point. Sibson's $Z^{(1)}$ interpolant smoothly blends between $Z^{(0)}$ and ζ with two weighting parameters α and β.

$$Z^{(1)}(\boldsymbol{u}) = \frac{\alpha Z^{(0)}(\boldsymbol{u}) + \beta\zeta(\boldsymbol{u})}{\alpha + \beta} \tag{2.9}$$

where

$$\begin{aligned}
\alpha &= \frac{\sum_i \lambda_i(\boldsymbol{u})d_i(\boldsymbol{u})}{\sum_i \lambda_i(\boldsymbol{u})/d_i(\boldsymbol{u})},\\
\beta &= \sum_i \lambda_i(\boldsymbol{u})d_i^2(\boldsymbol{u}),\\
\zeta(\boldsymbol{u}) &= \begin{cases} z_n & \text{if } \boldsymbol{u} = \boldsymbol{u}_n \\ \frac{\sum_i \frac{\lambda_i(\boldsymbol{u})}{d_i(\boldsymbol{u})}\zeta_i(\boldsymbol{u})}{\sum_i \frac{\lambda_i(\boldsymbol{u})}{d_i(\boldsymbol{u})}} & \text{if } \boldsymbol{u} \neq \text{any } \boldsymbol{u}_n \end{cases}\\
\zeta_n(\boldsymbol{u}) &= z_n + \boldsymbol{\nabla}_n^T(\boldsymbol{u} - \boldsymbol{u_n}),\\
d_n(\boldsymbol{u}) &= \sqrt{(\boldsymbol{u} - \boldsymbol{u}_n)^T(\boldsymbol{u} - \boldsymbol{u}_n)},
\end{aligned}$$

In the original paper $d_n(\boldsymbol{u})$ is simply the euclidean distance between the interpolation site and data site n, but Flötotto pointed out in her thesis that that choice is rather arbitrary and any smooth continuous function $f(\|\boldsymbol{u}-\boldsymbol{u}_i\|)$ can be applied as long as $f(\|\boldsymbol{0}\|) = 0$ [21]. She provides comparisons between distance functions of the kind $d_n(\boldsymbol{u}, k) = \left[(\boldsymbol{u} - \boldsymbol{u}_n)^T(\boldsymbol{u} - \boldsymbol{u}_n)\right]^k$ for values of $k \in \{0.25, 0.5, 1, 2\}$. A k value of 0.5 obviously recreates Sibson's original interpolant. Increasing the value k makes the β weight more dominant and the interpolant will more resemble a linear function in the vicinity of a data site. Decreasing the value of k has the opposite effect and the gradients at the data sites are of less influence. Setting $k = 0$ reduces the interpolant to a different interpolant Flötotto calls I^1 [21].

$$I^1(\boldsymbol{x}) = \sum_i \lambda_i(\boldsymbol{x})(z_i + \frac{1}{2}\boldsymbol{\nabla}_i^T(\boldsymbol{x} - \boldsymbol{p}_i) \tag{2.10}$$

Visual comparison gives the impression that this interpolant is of lower quality.
It can be shown that Sibson's $Z^{(1)}$ interpolant is able to reconstruct spherical quadratics exactly. While an interpolant defined by Farin can reconstruct any quadratic exactly by utilizing Bezier simplicies, will this interpolant not be considered in this work because of the increased computational time and implementation complexity for a very similar result as Sibson's $Z^{(1)}$ [26][27][28].

2.5.3.4 Ghost points

Ghost points are synthetic points added to the set of data points used to extend the domain of a natural neighbor interpolant as to effectively being able to use the interpolant as an extrapolant [29]. Extrapolation is the work of estimating function values outside of the domain of convex hull of the data points. The quality of the extended domain interpolant is dependent on the placement and number of ghost points. The ghost point methods can be divided into two main methods; dismissed ghost points and assigned ghost points.
Dismissed ghost point methods use the inserted ghost points in the calculation of the natural neighbor coordinates, but remove any coordinate referencing a ghost point and re-normalizes the rest of the coordinates. This method has the advantage that there is no need to assign values to the ghost points, but the resulting interpolant will only be C^0 at data points being natural neighbors of the ghost points.
Assigned ghost point methods generate feasible values at the ghost point using some method and proceed as in normal natural neighbor interpolation. Most global interpolation methods, such as IDW, are not limited to the convex hull of the data points and as such naturally extend into extrapolation without modification and can be used to assign ghost point values, but the quality of the extrapolant depends on the quality of the method of assigning values to the ghost points.

2.6 Bending Energy

The elastic bending energy of a thin metal rod with clamped ends is proportional to the integral of the squared curvature along the rod (2.11)[30].

$$E[\gamma] \propto \int_{\gamma} \kappa(s)^2 ds \tag{2.11}$$

In the case of bending energy of mathematical curves can the proportionality constant be defined as 1 and the proportionality of (2.11) can be exchanged for equality. This is known as the Euler-Bernoulli energy functional [31]. This measure increases with increasing length of the curve. By dividing the functional with the length of the rod, is a functional with units of force realized.

$$F[\gamma] = \frac{\int_{\gamma} \kappa(s)^2 ds}{\int_{\gamma} ds} \tag{2.12}$$

It can be thought of as the average energy per unit length or average bending force needed to deform a rod to shape of γ.

2.7 Machine Learning

Machine learning (ML) is a subfield of artificial intelligence (AI) where algorithms learn to approximate a function without explicit programming. This learning process typically involves optimizing a model's parameters based on observed data using statistical optimization techniques to minimize an error metric or maximize a notion of performance.
The machine learning in this thesis will be applied to an image-to-image regression task. There are multiple different algorithms that can be applied to this kind of task, but different approaches with artificial neural networks have by far been the most successful, and this is the approach that will be considered. Other possible approaches would include Markov random fields or decision forests applied per-pixel. All per-pixel approaches were excluded because of the cost of inference as the model would have to be applied to each pixel. Markov random fields were excluded because of the difficulty in defining an efficient potential function that would make the field converge towards the wanted output.

2.7.1 Artificial neural networks

Artificial neural networks are large statistical machinery inspired by the structure of the brain that have the ability to approximate any continuous function [32]. The basic computational unit is the artificial neuron. A neuron computes an affine transformation of its input vector $\boldsymbol{x}$ followed by a non-linear activation function σ. Specifically, the output scalar a is given by $a = \sigma(\boldsymbol{w}^T\boldsymbol{x} + b)$, where $\boldsymbol{w}$ is a weight vector and b is a scalar bias. a is called the activation of the neuron.
Efficient training of ANNs depends on differentiable activation functions to enable gradient-based optimization. Popular functions are the sigmoid, tanh, or the Rectified Linear Unit (ReLU).

In deep learning are multiple neurons organized in multiple sequential layers, which enables the network to learn hierarchical data representations from the input. This section outlines the theoretical basis of two fundamental architectures: the Multilayer Perceptron (MLP) and the Convolutional Neural Network (CNN).

MLPs organize the neurons in fully connected layers. Each layer works as a function $f : \mathbb{R}^m \to \mathbb{R}^n$ defined as $\boldsymbol{a}^{(l)} = \sigma^{(l)}(\boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)})$ where $\boldsymbol{a}^{(l)}$ is the activations of the n neurons in the layer. The inputs are the activations of the m neurons in the previous layer, $\boldsymbol{a}^{(l-1)}$, and a learned weight matrix $\boldsymbol{W}^{(l)}$ and bias vector $\boldsymbol{b}^{(l)}$. $\boldsymbol{a}^{(0)}$ is simply the input to the network, $\boldsymbol{a}^{(0)} = \boldsymbol{x}$.
MLPs with at least one hidden layer and appropriate non-linear activation functions are capable of approximating any continuous function to arbitrary precision given a sufficient number of neurons in the hidden layer [33]. The network parameters ($W^{(l)}$, $b^{(l)}$) are learned by minimizing a loss function, typically using gradient descent. Gradient descent computes a small update to the network parameters through differentiation of the loss function with respect to the network parameters. This involves a forward pass to compute a prediction, calculation of the loss and

backpropagation of the gradients via repeated application of the chain rule [34]. MLPs, being fully connected, have a large parameter count and do not take input topology into account, making MLPs inefficient for high-dimensional data with an inherent structure, such as images.

CNNs are specifically designed to utilize the structure of grid-like data[35]. CNNs employ learnable convolutional filters that are applied across the input volume to detect local patterns, regardless of position. The output of filter j at position $\hat{\boldsymbol{i}} = (i_1, i_2, \ldots, i_n)$ is given by $h_{\hat{\boldsymbol{i}}}^{(j)} = \sigma((\boldsymbol{W}_j * \boldsymbol{X})_{\hat{\boldsymbol{i}}} + b_j)$, where $*$ denotes convolution, $\boldsymbol{X}$ is the input volume of dimensionality n (typically 2 or 3 for grayscale or color images), $\boldsymbol{W}_j$ and b_j are the weights and bias of filter j.

The output of a single filter applied across the entire input is called a feature map, effectively being an activation map that indicates the presence of the specific feature learned by that filter across the input space. The weights within a single filter are shared across all spatial locations, drastically reducing the number of parameters compared to MLPs and efficiently representing translation invariance. Multiple filters are usually used per layer to enable the detection of many features.

CNN architectures often include pooling layers that reduce the spatial dimensions of feature maps. This reduces the parameter count while providing a degree of local translation invariance and merges semantically similar features into one. Chaining multiple convolutional and pooling layers allows the network to extract increasingly complex and abstract features. The resulting feature maps are typically flattened and followed by fully connected layers (as in MLPs) to produce the final output.

2.7.2 Supervised learning

Supervised learning trains models on labeled datasets, where each data point consists of an input and a corresponding output called a label. The objective is to learn a function that maps the inputs to the labels. Training often relies on minimizing a differentiable loss metric, which measures the discrepancy between the prediction and the label, typically using gradient-based optimization algorithms.

2.7.2.1 U-net

The U-Net is a CNN architecture entirely made of convolutional layers that is used for image-to-image mapping. The architecture was introduced as a tool for semantic segmentation in medical imaging by Ronnberger et al. in 2015 [36]. It employs an encoder-decoder structure with a contracting path to capture context and a symmetric expanding path for precise localization. Skip connections between corresponding encoder and decoder layers allow the network to combine high-resolution feature details with contextual information effectively, making it powerful for pixel-level classification common in supervised image analysis.

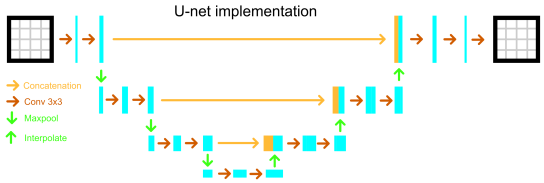


Figure 2.6: The implemented U-net architecture. One level shallower than the original U-net introduced by Ronnberger et al.

2.7.3 Unsupervised learning

Unsupervised learning tries to find patterns and structure in unlabeled data. Unlike supervised learning, there are no pre-defined output labels for training. The goal is typically to discover inherent clustering, reduce the dimensionality or learn underlying probability distributions or representations of the data.

2.7.3.1 Autoencoder

An Autoencoder (AE) is a common unsupervised neural network architecture used for dimensionality reduction. An AE learns compressed data representations by training a bottlenecked network to reconstruct its input [37]. It consists of an encoder $\boldsymbol{h} = g(\boldsymbol{x}; \boldsymbol{\theta})$ that maps the input $\boldsymbol{x} \in \mathbb{R}^n$ to a latent space $\boldsymbol{h} \in \mathbb{R}^q$, and a decoder $\hat{\boldsymbol{x}} = f(\boldsymbol{h}; \boldsymbol{\phi})$ that reconstructs $\boldsymbol{x}$ from $\boldsymbol{h}$.

AEs are trained by minimizing the reconstruction error. AEs learn compressed representations or features directly from the data structure and are useful for dimensionality reduction, feature extraction, denoising, and generative modeling. These learned representations can subsequently be utilized in other ML tasks, such as initialization or feature input for supervised or reinforcement learning models.

2.7.4 Reinforcement learning

Reinforcement learning (RL) is a ML paradigm in which an agent learns optimal behavior by interacting with an environment. RL operate without labeled data, instead using the environments reward signal to guide learning towards desirable outcomes. The agent selects actions based on the current state, receives a reward and transitions to a new state. The standard RL objective is to learn a strategy for action selection, called a policy, that maximizes the expected cumulative reward over time, through interaction with the environment.

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\rho_\pi} \left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right] \tag{2.13}$$

Maximum Entropy RL augments the objective to also maximize policy entropy [38].

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\rho_\pi} \left[\sum_{t=0}^{\infty} \gamma^t (r(\boldsymbol{s}_t, \boldsymbol{a}_t) - \alpha \sum_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s}_t) \log \pi(\boldsymbol{a}|\boldsymbol{s}_t)) \right] \tag{2.14}$$

where α is the temperature parameter balancing between optimizing for the reward and for the stochasticity of the policy.
This maximum entropy objective encourages exploration and improves the stability and robustness of learning.

2.7.4.1 Soft Actor Critic

Soft Actor-Critic (SAC) is an off-policy RL algorithm designed for continuous action spaces. It learns a stochastic policy $\pi_\phi(\boldsymbol{a}|\boldsymbol{s})$ (actor) and an ensemble of soft Q-functions $Q_{i,\boldsymbol{\theta}_i}(\boldsymbol{s},\boldsymbol{a})$ (critic) to maximize the RL entropy objective (2.14)[38]. SAC utilizes MLPs for the actor and critic.

The soft state value function is defined as $V(\boldsymbol{s}_t) = \mathbb{E}_{\boldsymbol{a}_t\sim\pi}[Q(\boldsymbol{s}_t,\boldsymbol{a}_t) - \alpha\log\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)]$. Each Q-function is trained by minimizing the soft Bellman residual using transition tuples $(\boldsymbol{s},\boldsymbol{a},r,\boldsymbol{s'})$, called experiences, from a replay buffer $\mathcal{D}$ (2.15).

$$\begin{aligned} J_Q(\boldsymbol{\theta}) &= E_{(\boldsymbol{s},\boldsymbol{a},r,\boldsymbol{s'})\sim\mathcal{D}}\left[(Q_{\boldsymbol{\theta}}(\boldsymbol{s},\boldsymbol{a}) - y)^2\right] \\ y &= r + \gamma\mathbb{E}_{\boldsymbol{a'}\sim\pi_\phi}[Q_{\bar{\boldsymbol{\theta}}}(\boldsymbol{s'},\boldsymbol{a'}) - \alpha\log\pi_\phi(\boldsymbol{a'}|\boldsymbol{s'})] \end{aligned} \tag{2.15}$$

where $\bar{\theta}$ are parameters of the corresponding target Q-network, updated via Polyak averaging $\bar{\boldsymbol{\theta}} \leftarrow \tau\boldsymbol{\theta} + (1-\tau)\bar{\boldsymbol{\theta}}$. The policy π_ϕ is trained to maximize the expected soft value, corresponding to minimizing equation (2.16).

$$J_\pi(\boldsymbol{\phi}) = E_{\boldsymbol{s}\sim\mathcal{D},\boldsymbol{a}\sim\pi_\phi}[\alpha\log\pi_\phi(\boldsymbol{a}|\boldsymbol{s}) - Q_{\boldsymbol{\theta}}(\boldsymbol{s},\boldsymbol{a})] \tag{2.16}$$

where a is sampled from the distribution generated by the policy using the reparameterization trick ($a = f_\phi(\epsilon;\boldsymbol{s})$ where ϵ is noise) to allow gradient backpropagation.
SAC can also learn the temperature α by optimizing a dual objective to meet a target entropy constraint.

$$J(\alpha) = \mathbb{E}_{\boldsymbol{a}\sim\pi_\phi}[-\alpha\log\pi_\phi(\boldsymbol{a}|\boldsymbol{s}) - \alpha H_{target}] \tag{2.17}$$

Multiple Q-functions are used to reduce the bias in the state value estimation by taking the minimum output out of all the Q-functions [39]. SAC is often applied to robotic tasks where environment steps can be costly, to address this does SAC allow multiple update steps per environment interaction by sampling from the replay buffer. Several strategies have been employed to further increase the sample efficiency. REDQ is a SAC variation that uses a high update to data (UTD) ratio. High UTD ratios lead to overestimation of the Q-values, which REDQ handles by employing a large ensemble of N (on the order of 10) Q-functions to reduce the bias. While REDQ is highly sample efficient, is the need for gradient updating of N Q-functions expensive. DroQ addresses this by only using M ($M < N$, often 2) Q-functions, but adds dropout regularization to the MLPs to increase the variance and reduce the overestimation bias [39].

2.7.4.2 Soft Actor Critic Autoencoder

Soft Actor Critic Autoencoder (SAC-AE) integrates an Autoencoder with the SAC algorithm, primarily to handle RL tasks with high-dimensional state spaces, such

as images. The AE is embedded into the soft actor critic architecture and learns a compact latent representation of the raw observations along side learning the policy [40]. The deterministic autoencoder $\hat{\boldsymbol{o}}_t = f_{\boldsymbol{\eta}}(\boldsymbol{z}_t),\ \boldsymbol{z}_t = g_{\boldsymbol{\psi}}(\boldsymbol{o}_t)$ is trained with a regularized reconstruction loss.

$$J_{RAE}(\boldsymbol{\psi}, \boldsymbol{\eta}) = \mathbb{E}_{\boldsymbol{o}_t \sim \mathcal{D}} \left[\log p_{\boldsymbol{\eta}}(\boldsymbol{o}_t | \boldsymbol{z}_t) + \lambda_z \|\boldsymbol{z}_t\|^2 + \lambda_{\boldsymbol{\eta}} \|\boldsymbol{\eta}\|^2 \right] \tag{2.18}$$

The SAC agent's actor and critic operate on these low-dimensional latent representations z_t instead of the high-dimensional inputs.

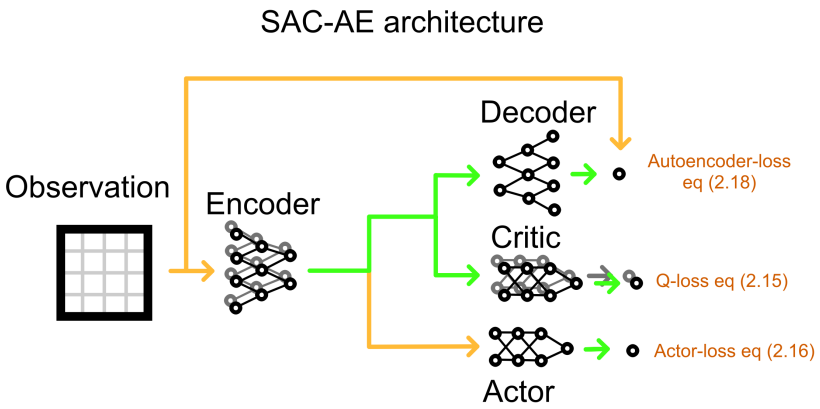


Figure 2.7: The soft actor critic architecture. The signal is propagated forwards by the yellow and green arrows, but the gradients are only propagated backwards along the green arrows.

A crucial stabilization technique is blocking actor gradients from updating the encoder $g_{\boldsymbol{\psi}}$. The encoder parameters $\boldsymbol{\psi}$ are updated only by gradients from the critic loss (2.15) and the AE loss (2.18). The actor $\pi_{\boldsymbol{\phi}}$ is updated using (2.16). The decoder $f_{\boldsymbol{\eta}}$ is updated by (2.18). Additionally, asymmetric target network updates with different Polyak coefficients ($\tau_{enc} > \tau_Q$) are used for the encoder weights ($\bar{\boldsymbol{\psi}}$) and the critic parameters ($\bar{\boldsymbol{\theta}}$) in the target Q-network to aid signal propagation to the encoder[40].

3
Methods

Minimizing the information loss in contour extraction from DEMs while the resulting contours are constrained to being well behaved (easily readable, C^1 smooth and representing possible geometry i.e. no contours are crossing each other) is a complex problem. The first step is to define a scoring function S that measures the descriptive power of a contour set.
By assuming the DEM models a C^1 function, the isolines, being slices of the function, will be C^1 curves (or points, but that will not be the case in this work because of the use of marching squares for contour extraction).
Working purely on the isolines by trying to generalize them cannot work, as too much information about the landforms in between the isolines is lost in the contour extraction. It would also easily lead to issues with contour geometry if one is not careful.
A hybrid approach where one generates isolines and adjusts them based on the DEM is a possible approach, but it is hard to formalize rules for how to adjust the contours and when to create a new contour or when to remove a contour.
A DEM based approach guarantees geometrically valid contours. It assumes that the contour set that retains the most information from a DEM can be extracted as the isolines of another DEM. This assumption is the basis of DEM based approaches and makes it possible to represent a contour set by a DEM for which the contour set can be retrieved by extracting the isolines. This method will be described in this section.

The scoring function will be defined in the form of penalized least squares. Penalized least squares is a regression method that aims to find the best fit for a set of data while also penalizing the complexity of the model [41]. In our case this will take the form of a weighted sum of the mean squared error of the DEM interpolated from a contour set with respect to the true DEM penalized by a measure of the squiggliness of the contours.
Using the DEM assumption, the contour set will be represented by a DEM and a fixed contour extraction function.
The scoring function S can mathematically be defined by:

$$S_\lambda(A|T) = \|T(x,y) - I(C(A(x,y)))\|_2^2 / N^2 + \lambda \sum_i F(C(A(x,y))_i) \tag{3.1}$$

where:
$T : \mathbb{R}^{[N\times N]}$ is the true DEM,
$A : \mathbb{R}^{[N\times N]}$ is a DEM extracted from a contour set,
$C : \mathbb{R}^{[N\times N]} \to \mathcal{C}$ is a function that extracts a contour set from a DEM,

$I : \mathcal{C} \to \mathbb{R}^{[N \times N]}$ is an interpolation function which interpolates a DEM from a contour set,
$F : \mathcal{C} \to \mathbb{R}$ is a function that outputs a measure of the squiggliness of a contour,
$\lambda : \mathbb{R}$ is a weighting parameter.

This chapter will walk through the process of defining the different terms of equation (3.1). First, the DEM format will be defined and the process of deriving them from lidar point clouds will be explained. Secondly, will the function C for extracting contours from a DEM be selected. After that will the interpolation function I be chosen, inspired by how experienced map reads interpret contour maps. The function F that outputs a measure of the squiggliness of a contour is then defined. With these functions defined, is the objective scoring function for a contour set $S_\lambda(A|T)$ defined. With the measure for the goodness of a contour set defined, will optimization techniques be investigated in the next chapter.

Every section in this chapter is divided in two. The first part is about the reasoning behind the chosen approach and the application of each algorithm and the second part is about the implementation details for the various algorithms.

Programming language

The language chosen for all programming related tasks was Rust. Rust is a low-level statically typed compiled programming language focusing on memory safety and performance [42]. Rust was chosen for its speed and the large ecosystem of libraries available. Both Python and C++ could also have been chosen, with their even larger ecosystems, but they were decided against because of Python's slow computation speed and because the author enjoys writing Rust more than C++.

Notable libraries used

The Burn deep learning framework was used in the machine learning approaches described in later chapters. Burn is a pure Rust deep learning framework that emphasizes performance, flexibility, and portability for both training and inference [43]. Burn was chosen over the more established Python and Pytorch because the objective function (3.1) is quite expensive to compute.

Eframe was chosen as the application framework. An advantage with Eframe is that the code is not platform dependent and the Rust build system makes it easy to cross compile the application to a great number of platforms, even the web through web assembly[44].

The Geo library provides multiple robust and efficient vector geometry algorithms used in this work[45].

Spade was used for triangulation and interpolation and was used as the basis of the implementation of the interpolation function in (3.1)[46].

3.1 DEM generation

DEMs are rasterized representations of elevation functions. Today they are most often derived from lidar point clouds [2]. Although most lidar providers also provide DEMs, we chose to derive our own DEMs from lidar data. The main reason for this is to be able to control the resolution of the DEM regardless of the provider. The process of gridding lidar data can be memory intensive for large point clouds and precautions have to be taken to not run out of memory.

The DEM resolution was set to 0.5 meters, as it is half of 1 meter, which is smallest feature size that orienteering maps depict. This matters because of the Nyquist-Shannon sampling theorem, which states that the sample rate needs to be at least double the frequency of a signal to measure the signal without aliasing effects occurring [47].

The next important parameter is the DEM side lengths. Gridding the entire extent of the point cloud to a raster and handing the entire raster to the contouring functions has its advantages. This leads to few tiles, and thus few areas where tile boundary problems can occur, and might also be decently efficient for the simplest contouring algorithms. However, the disadvantages are greater. The first issue is the memory consumption, any gridding algorithm relying on a triangulation of the lidar points will have memory demands at least proportional to the number of points. An even greater issue is the dimensionality of the domain and codomain of a function that maps a DEM to DEM (3.2).

$$f : \mathbb{R}^{[N\times N]} \to \mathbb{R}^{[N\times N]} \tag{3.2}$$

where N is the side length of the DEM. Large domains and codomains make many machine learning based contouring algorithms prohibitively expensive. However, for the simplest functions that only operate on small neighborhoods of the DEM at a time does the size of the domains not play a role and larger DEMs are advantageous.

For this thesis was the chosen DEM size 256 pixels by 256 pixels or 128 meters by 128 meters, with an overlap between neighboring tiles of 14 meters. Although supervised learning techniques should be able to handle larger tiles without demanding an unreasonable amount of data, is 256 by 256 pixels a large, but manageable, state space for reinforcement learning techniques [40]. The DEM size was kept consistent across all techniques for comparability and thus had to be limited in size to enable reinforcement learning techniques.
The ground and water classified lidar points were interpolated using Sibson's $Z^{(0)}$ interpolant, see equation (2.5). The choice of using the $Z^{(0)}$ interpolant was made because it is the highest quality local scattered data interpolant when gradients are not available. As $Z^{(0)}$ is only defined in the interior of the triangulation, was ghost points added to each of the four corners of the DEM tile to make the entire DEM interpolatable. The ghost points were created through IDW interpolating the closest four points in the point cloud for each corner of the DEM. If the DEM corner coordinates are in the interior of the point cloud, then the new points will have little

effect on the interpolated DEM. On the other hand, if the DEM corner was outside the point cloud's convex hull, which often happens for DEM tiles with a corner in a lake (as water reflects little lidar laser energy) or on tiles on the boundary of the lidar project, then the ghost points will extend the domain of the interpolant to cover the entire area of interest. Ghost points far from the neighboring points will converge towards the average value of the neighbors, this makes the elevation field gradually flatten past the convex hull of the point cloud, which is a wanted behavior as most reasons for the need of ghost points, apart from the boundary of the lidar project, are bodies of water which reflect little of lidar pulses.

3.1.1 Implementation

The input to the application are lidar files in the LAS format or its compressed equivalent LAZ. Lidar files are large files containing point clouds, where each point is tagged with data [48]. The exact data depend on the LAS version and point record format, but all lidar files have the data fields which interest us, namely x, y and z coordinates and a classification field. The classifications are labels added in postprocessing of the point cloud. Most aerial lidar providers classify the points into the classes; ground points, low noise, high noise and unclassified. Many providers also classify water points, but only on large bodies of water and with varying accuracy. The classes that are interesting in DEM generation are ground and water points, which we will treat equally. Every input lidar file is assumed to be classified with at least ground points. The point coordinates are in LAS-files stored as 64-bit signed integers which are converted to 64-bit floating point coordinates by scaling and offsetting by the six 64-bit transformation floats stored in the files header data, two for each spatial dimension. All modern LAS readers do this conversion automatically when reading the points from the file. For most aerial metric lidar data is the scaling a conversion from centimeters to meters, which limits the resolution to at most one centimeter, but that is smaller than accuracy of most lidar sensors and so it does not represent any significant data loss. Every lidar file is required to contain a record describing the coordinate reference system (CRS) the coordinates are relative to. The CRS can be given by the Well Known Text (WKT) definition or in the legacy GeoTiff format depending on the LAS specification version.
Each DEM was generated as a tile of 128 by 128 meters or 256 by 256 pixels, with an overlap to the neighboring tiles of 14 meters. The small tile size was chosen to limit the input size of the contour optimization algorithm. The issue with small tile sizes is the increasing number of tile edges and the continuity issues associated with them. An overlap of 14 meters between neighboring tiles was found sufficient to mitigate boundary issues when extracting isolines. Continuity across tile edges cannot be guaranteed if optimization algorithms are applied to the tiles without any sort of communication between neighboring tiles.

Aerial lidar files cover large areas, often ranging from 0.5 to 10 square kilometers, and with a point densities of 0.5 to 30 points per square meter, a single file can contain tens of millions of points. Filtering the cloud for ground points significantly reduces the amount, but still trying to triangulate and interpolate the remaining

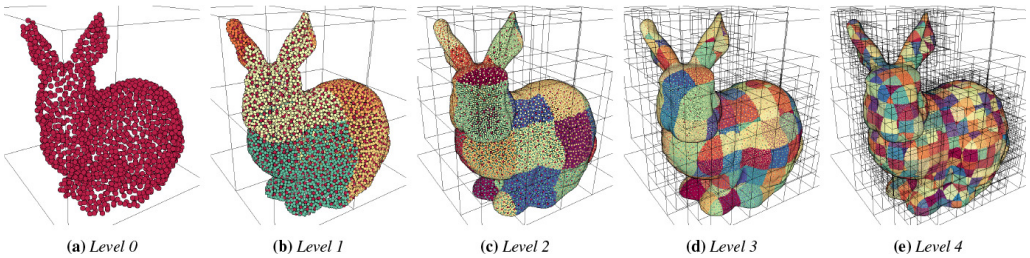


Figure 3.1: A rabbit shaped point cloud stored in an octree. Each level subdivides the space in 8 and increases the capacity of the level by a factor of 8. Each level is a lower resolution representation of the whole point cloud, with increasing density as the level number increases. The union of all levels is the entire point cloud.

points directly will quickly lead to memory issues for the larger files on an average laptop computer. This does not even take into account that to avoid edge issues must the closest points from the neighboring files also be included in the triangulation, further worsening the problem. Obviously some spatial filtering must be performed to divide the points into smaller more manageable clouds. The naive approach would be to loop through every point in both the current cloud and the neighboring clouds and distribute the ground points to the correct tiles. This is both compute intensive and requires either holding every ground point in memory at once or writing intermediate files, both of which is slow for large point clouds on normal laptop computers.

A better approach would be to have the point cloud organized in a data structure which facilitates quick spatial look-ups. One such data structure is called an *octree*. An octree is a tree structure in which each node has eight children. This is convenient for spatial data in three dimensions as the levels of the tree can be used to recursively subdivide three dimensional space into octants. By moving one level down in the octree, the node boundaries' side lengths are halved, but the capacity for points in the level increases by a factor of eight, see figure 3.1. Spatial queries on octrees can be done in logarithmic time as entire sub-trees can be excluded from the search if a parent node falls outside the search area.

Two such formats where compressed lidar points are stored in an octree are Entwine Point Tile (EPT) and Cloud Optimized Point Cloud (COPC) [49]. EPT is an exploded file structure where every octree node is stored as its own compressed lidar file. The octree hierarchy information and lidar file metadata are stored in JSON files. COPC is a compressed LAS format that follows the LAS 1.4 specification with an embedded octree structure modeled after the EPT structure [49]. Advantages with COPC over EPT include smaller total file sizes, slightly faster spatial queries and easier file management as a single large file is easier to keep track of than many small files. All normal compressed LAS 1.4 readers can read COPC files, but to take advantage of the internal octree structure, a specialized reader is needed.

Compressed lidar files compress the points into chunks using an arithmetic compression algorithm [50]. The compression of a chunk does not depend on any other chunk and the chunks can therefore be read in any order. However, decompression of the points inside a chunk must be done sequentially. To read the last point in a chunk, all the points in the chunk need to be decompressed. One LAZ file consists

of a header, multiple chunks and a chunk table to keep track of the byte offset where in the file the different chunks start in addition to other uncompressed records such as a CRS record. The octree structure is embedded in the compressed lidar file by compressing all points belonging to a single octree node into a single LAZ chunk. The chunks of COPC files are often smaller than those of unordered LAZ-files, leading to slightly larger files, but only by a factor of about 10% depending on the node size. COPC stores one additional table, compared to unordered LAZ-files, which map chunks to nodes in the octree, and one extra record, which can be considered the octree header, which describes the root node's spatial boundaries and byte offset in the file.
The first step of DEM creation is to convert every non-COPC lidar file to COPC files. After that, the lidar files' spatial relationships are identified and the DEM extents, both including and excluding the overlaps, are calculated. The DEMs are created by querying the COPC file for the DEM extent, if the DEM is on the edge of a lidar file then even the neighboring lidar file or files are queried. The result is a small point cloud that can be triangulated and interpolated without worrying about running out of memory.
After generating the DEMs the next step is extracting isolines which can be extracted using a contouring algorithm. The algorithm chosen for this purpose is called Marching Squares.

3.2 Contour extraction with Marching Squares

Marching squares extract directed contours along an isolevel by walking through the raster grid one square at a time and linearly interpolating the corner values of the square. The algorithm was chosen as the base contour extraction algorithm used in every extraction operation in this thesis as it is the most widely utilized contour extraction algorithm for raster data. While more efficient contour extraction algorithms exist, marching squares is still very efficient for the size of rasters used in this work. Marching squares is also easy to implement. The problem with two different possible sets of line segments for configuration 5 and 10, as discussed in the theory chapter, is solved by always choosing the line segments going along the top-left to bottom-right diagonal, the brown arrows in figure 2.1. The reason behind this choice was to keep the implementation of the algorithm as simple as possible as this choice minimizes the number of contour merging operations which also leads to slight performance improvements.
The contour extraction algorithm will be executed quite often in the process of creating contours, as such is the implementation of marching squares an area where slight improvements pay off.

3.2.1 Implementation

A common way to implement the algorithm is to first calculate all line segments by marching through the grid, and afterward assemble the segments tip-to-tail into contours. A more efficient approach would be to assemble the contours on the fly. This can be done by realizing that the type of contour extension is entirely given by

the configuration of the corner values with respect to the isolevel.
By giving each corner an index and indexing into a lookup-table (LUT) based on the configuration given by $\sum_{i=0}^{3}(c_i > s) * 2^i$, for an isolevel s and the corner value c_i at the corner index i, is what edges that should be crossed by the directed segment retrieved, see figure 2.1.
As a side effect of marching the square through the grid row-by-row, will the different configurations also give in what way a segment should be added. This is not true at the boundary of the grid, but that can be mitigated by padding the grid with one cell on every side. The padded cell values should be all above or below the isovalue to ensure that no line segment will cross the boundary of the grid. The padded value is set to be the minimal value that a double precision floating point can store. This has the side effect of closing every contour which, while not relevant for polyline extraction, is great for extracting vector polygons.

With corner indices as in figure 2.1 are the additions as a function of the configuration given by:

$$\begin{aligned}
&\text{configuration} \in \{0, 15\} \implies \text{do nothing} \\
&\text{configuration} \in \{1, 14\} \implies \text{merge two contours (or close a contour)} \\
&\text{configuration} \in \{2, 3, 6, 7\} \implies \text{append segment to existing contour} \\
&\text{configuration} \in \{8, 9, 12, 13\} \implies \text{prepend segment to existing contour} \\
&\text{configuration} \in \{5\} \implies \text{prepend the first and append the other} \\
&\text{configuration} \in \{10\} \implies \text{append the first and prepend the other}
\end{aligned}$$

Adding in the observation that segments only are added along the active edge, and keeping a map of what contour is ending at what part of the active edge, is both which contour the segment should be added to and what kind of addition that should be made given, see figure 3.2. This also has the advantage of not being susceptible to floating point inaccuracies, which other approaches using coordinates for matching the contours to the correct segment have.

The entire DEM can be fitted into an array and kept in stack memory for fast access by further restricting the raster data structure to be of a fixed sized, $N \times N$.
The length of the active edge is $N + 2$. A map from the active edge segments to the relevant contour can be stored as an array of length $N + 2$. Each horizontal segment has, from left to right, the indices 0 to N and the moving vertical segment has index $N + 1$, see the blue line in figure 3.2. The map is updated when a new segment is added by writing to which contour the segment was added in the map at the current index or in the vertical index, depending on which configuration was calculated. A small detail worth noticing is that when no segment is added (due to all four corners of the square being either above or below the isolevel), then the map does not need updating. This leaves the map in an incorrect state, but the error is guaranteed to be corrected before the bad map entry is read. This is a consequence of the marching squares algorithm processing every edge in the grid twice (except the boundary edges, but these edges are guaranteed to be ignored due to the padding

of minimal values, effectively removing any edge cases from the algorithm). The first pass either writes to the map or ignores the edge, and the second pass can only read the entry if the first pass wrote to the map. Some consideration must be taken when merging contours, given by configurations 1 and 14, as this will change the name of one of the merging contours as well as affect the last contour which will fill the space of the contour that was merged, see figure 3.2, the map must be updated to reflect the name changes.
While the fact that every edge is processed twice simplifies the implementation, it is also an obvious area for computational optimization. Faster algorithms with no repeating of edge processing exist [51]. However, since most edges will not be crossed by a contour, balances the current implementation speed and complexity of implementation.

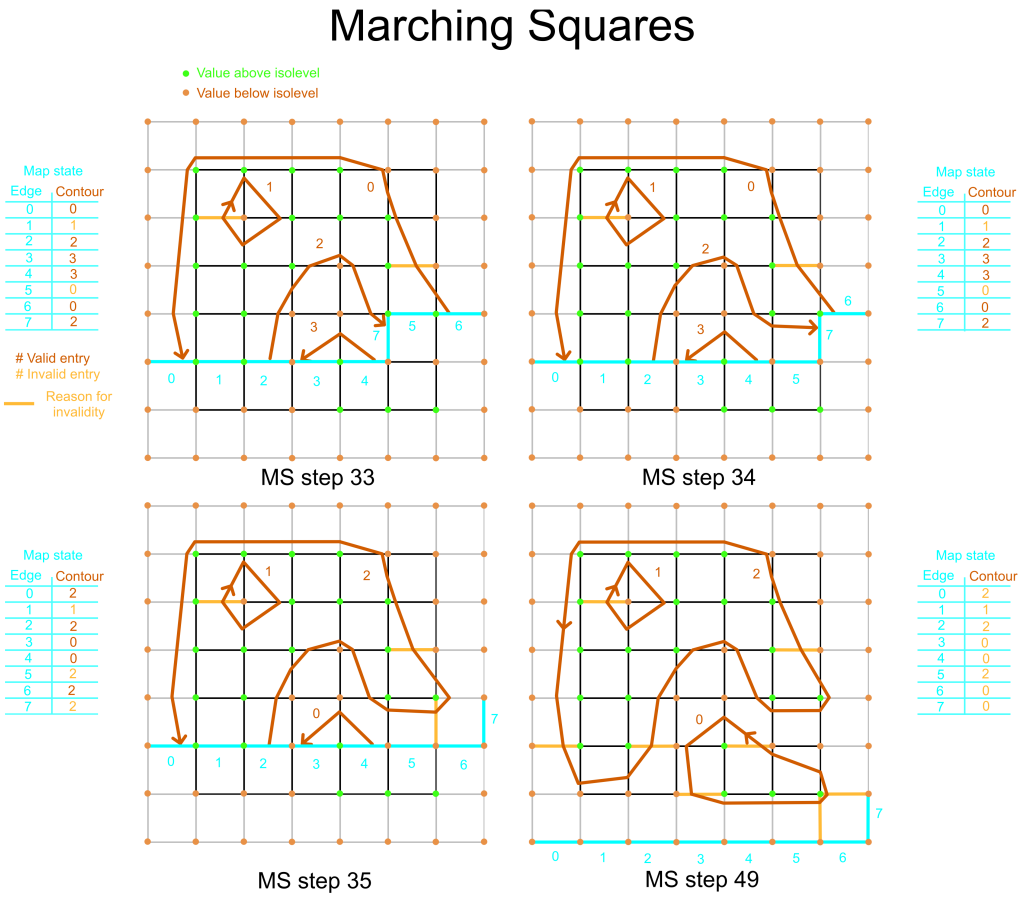


Figure 3.2: The marching squares algorithm on a 6 by 6 grid padded by 1 cell on every side. The padded values are all below the isolevel. This removes all edge cases as no line segment can cross the outer edge and also closes all contour lines. The blue line is the active edge, were line segments can be added and an array is kept mapping from the active edge to the contours. In step 35 is contour 2 and 0 merged by adding 0 to the end of 2. This changes the address of contour 3 to 0 and all values in the map referencing both contour 3 and 0 are updated.

3.2.2 Simplification

The marching squares algorithm on a grid of resolution 0.5 meters generates a contour vertex every 1/2 to $\sqrt{2}/2$ meters along the contour. Most of these vertices are redundant, and some simplification that preserves the contour shape should be performed to reduce the data size and simplify the next steps of the process. The chosen algorithm for this purpose was Ramer-Douglas Peucker (RDP) simplification algorithm.
RDP simplification is a polyline simplification algorithm that produces simplified lines with the least amount of disparity to the input given a maximum allowed disparity at any one point. While the algorithm is more time consuming than other algorithms, such as Zhao-Saalfield, it was considered fast enough [18]. The simplification distance used in simplifying the raw marching squares output was 0.1 meters. The value was found small enough to not significantly change the contour shape, while still reducing the vertex count drastically. The rust Geo library provided the implementation of RDP.

The function $C : \mathbb{R}^{[N\times N]} \to \mathcal{C}$ from equation (3.1) was defined as Marching squares followed by RDP simplification with $\epsilon = 0.1$m.

After having applied RDP to the extracted isolines, are the isolines ready for back-interpolation into a DEM.

3.3 Interpolation

The point of the interpolation step is to try and generate a DEM from a contour set in the same fashion that experienced map readers derive elevation from contours. After having discussed this with some elite orienteers from IFK Göteborg Orientering four rules for the interpolation algorithm were identified.

1. Local support, contours far away do not affect the interpretation of a feature.
2. Continuous non-linear surface, the area in between contours of the same height is expected to either dip down and then rise up or the opposite, not stay at the same level or have any jumps.
3. Proximity to other contours matters, higher contour density means steeper hills.
4. In the case of lone contours, there is expected to be some steepness with a minimum magnitude depending on the terrain type.

Rule 1 eliminates all global interpolation schemes.
Rule 2 eliminates terrace interpolation, linear interpolation and nearest neighbor based interpolation. From rules 3 and 4 it is clear that the interpolation scheme should include gradients derived from the neighboring contours clamped with a minimum magnitude. Finally, as the contour data are irregular, only scattered data interpolants can be applied.
From these rules, three possible interpolants were identified. Sibson's C^1 interpolant, Farin's C^1 interpolant and Hiyoshi C^2 interpolant. Hiyoshi's interpolant was eliminated because of complexity and the need for deriving the Hessian [21]. Farin's and

Sibson's C^1 interpolants are quite similar [27]. They are both based on Sibson's natural neighbor coordinates derived from a Delaunay triangulation of input data points and both depend on the gradients at the data points. Farin's interpolant is slightly stronger than Sibson's, as it can reproduce a general quadratic polynomial without error, while Sibson's only can reproduce spherical quadratics [20]. However, Farin's interpolant adds complexity and computation time and in practice are the interpolation qualities of the two methods very similar. Sibson's C^1 interpolant was chosen for those reasons.

3.3.1 Implementation

Similar to the Delaunay triangulation and Sibson C^0 interpolation in the DEM creation, the Spade rust library was used for triangulation and Sibson C^1 interpolation of the contours. Sibson's $Z^{(1)}$ interpolant had a small error in the library, which was quickly identified and corrected.

The gradient directions at the data points are given by the fact that they are perpendicular to the contour lines. The magnitudes had to be derived. Sibson's gradient estimation method could have been implemented here, but that felt like overkill since we only need the magnitude and not the entire gradient. Sibson's gradient estimation involves inverting a $k \times k$-matrix, where k is the number of natural neighbors of the current vertex, for every vertex of the contour lines. For a two-dimensional Delaunay triangulation is k at least 3 and on average 6. While not very large is this still expensive to compute for every vertex and a simpler strategy was chosen.

For each natural neighbor, the slope of the vector from the current data point to the neighbor was calculated. This slope was dotted with the known direction of the gradient at the data point which gives a measure of the slope in the direction of the gradient. The maximum calculated slope in the direction of gradient out of all of the natural neighbors was kept, but clamped by a minimum value to respect Rule 3.

$$\begin{aligned} l_i &= \max_{k \in N_i} \left\{ \frac{\boldsymbol{u}_{ki}^z}{\|\boldsymbol{u}_{ki}\|_2^2} (\boldsymbol{u}_{ki}^x, \boldsymbol{u}_{ki}^y) \cdot \tilde{\boldsymbol{\nabla}}_{\boldsymbol{i}} \right\}, \quad \boldsymbol{u}_{ki} = \boldsymbol{u}_k - \boldsymbol{u}_i \\ \boldsymbol{\nabla}_{\boldsymbol{i}} &= \max\{l_i,\ s\} \tilde{\boldsymbol{\nabla}}_{\boldsymbol{i}} \end{aligned} \tag{3.3}$$

Where $\tilde{\boldsymbol{\nabla}}_i$ is the direction of the gradient $\boldsymbol{\nabla}_i$ at contour vertex i.

The reason for taking the maximum directional slope value instead of some averaging approach is to preserve as much information about the nearby slopes as possible.
Similarly to the DEM interpolation does the contour interpolation have an issue with the domain over which the interpolant is defined. In most cases will the contour set not contain vertices in all four corners of the DEM that should be interpolated. Ghost points was again employed to solve this issue. One ghost point was placed in each corner with its value taken from DEM that the contours were extracted from. This pins the corner values of the interpolated DEM to the corner values of the

generating DEM, but was seen as the best approximation of the true corner value.

Sibson's $Z^{(1)}$ interpolant with clamped gradients and ghost points was the function decided on for $I : \mathcal{C} \to \mathbb{R}^{[N \times N]}$ from equation (3.1).

3.4 Contour energy

A measure of how squiggly a contour had to be defined, this measure will be called the contour energy. The chosen measure was inspired by the energy needed to elastically deform a stiff rod with clamped end points. This energy is, up to a material constant, given by the Euler-Bernoulli bending energy functional given by equation (2.11). The bending energy is the most common penalty term used in penalized least squares methods.

However, the bending energy is not directly useful for defining a measure of the squiggliness of contours. This is because longer contours will lead to larger energy values, while the squiggliness is not necessarily larger. To get a measure better suited for contour energy was the bending energy normalized by dividing by the curve length, given by equation (2.12). This gives a quantity with units of force and can be interpreted as the average bending force needed to deform a thin elastic rod to the shape of the contour. This is a better measure for the squiggliness of a contour as we do not want to punish long contours.

With $F : \mathcal{C} \to \mathbb{R}$ defined was every term in (3.1) defined and the scoring function completed. The next step was to use the function to optimize a contour set.

3.4.1 Implementation

The measure of bending force was implemented by iterating over every contour in the contour set and summing the numerically calculated curvature of each contour at every vertex divided by the average of the neighboring segment lengths.

4

Optimization

Having defined the scoring function (3.1) is the next step to try and use it for optimization.
The optimization task is defined by equation (4.1).

$$A^* = \arg\min_{A(x,y)} \{S_\lambda(A|T)\} \tag{4.1}$$

Equation (4.1) defines the optimal DEM, A^*, for extracting isolines that describe the true DEM, T, with respect to the scoring function S_λ. A lambda value of 0.01 was found to make the contributions of the error and squiggliness term roughly equal and was kept constant across all methods.

The main problem with optimizing the function (3.1) is that it is not differentiable. This excludes all gradient based approaches.

4.1 Iteration based optimization

The first family of algorithms to be applied to the optimization problem (4.1) are iteration based. The idea is that successive small improvements are easier to calculate than one-shotting the optimal DEM. The framework on which all the approaches are based is depicted in figure 4.1. Notably, the framework has two functions that can be swapped out; the initialization function and the update function.
The point of the initialization function is to try and get as close to an acceptable

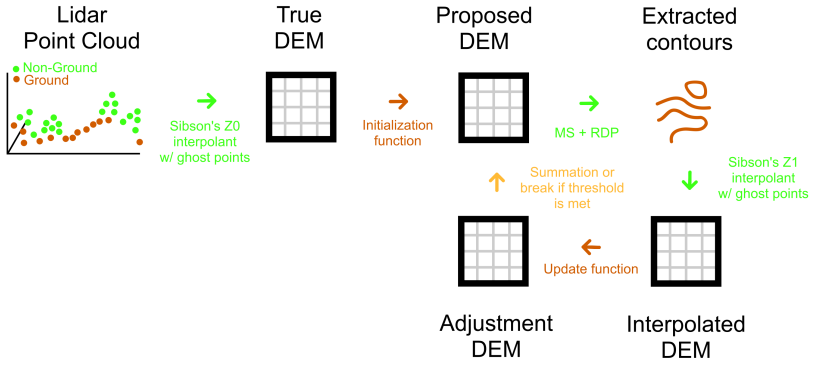


Figure 4.1: The green arrows represent fixed functions. The brown arrows represent functions that are swapped between methods.

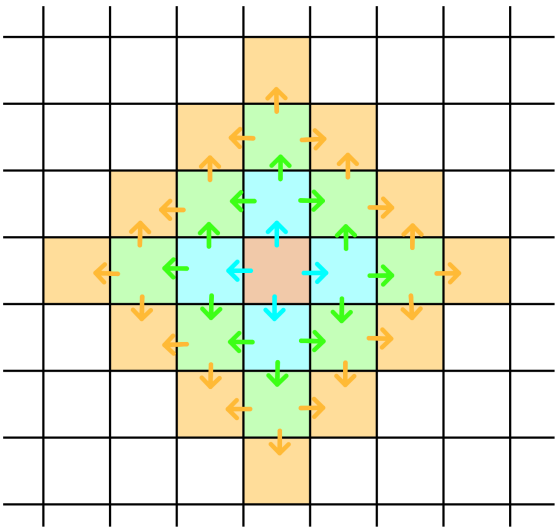


Figure 4.2: Contour continuity across tile boundaries can be enforced by starting from a central tile and in each step move outwards and calculate the contours for all 4-neighbors, while freezing the overlap regions from previously calculated tiles.

solution as possible before starting the iterations. Two initialization functions were tested; the first one was simply the identity function and the other was normal field smoothing.

4.1.1 Normal field smoothing

Normal field smoothing is an adaptive DEM smoothing technique that works on the normal field of the DEM instead of on the DEM values directly [6]. The algorithm has a couple parameters such as the number of update iterations, filter size and the threshold for how diverging neighboring normal vectors have to be to not smooth each other.

The filter size was set to 15 grid cells and the smoothing threshold to 15 degrees as recommended by Lindsay et al. [6]. The number of elevation update iterations was left as an adjustable parameter exposed in the graphical user interface.

4.1.2 Continuity across tile borders

Keeping the contours continuous across the tile borders is an important task any contouring algorithm must do. This can easily be done in the iteration framework by starting on a central tile and spreading out according to the figure 4.2. Overwriting the next iterations DEMs on the shared edges by the already optimized tile's edge values and freezing those values from the iteration updates. A smooth step function should probably be used for weighting the neighboring DEMs in the areas close to the overlap region to avoid any sudden jumps in elevation values.

4.1.3 Naive update

The first optimization technique is the simple naive approach. The adjustment DEM in each iteration is just the error for each pixel in the interpolated DEM with respect to the true DEM. Both the identity and Normal field smoothing was tested as initialization functions.
This has the disadvantage of not incorporating the energy of the contours in the update function. To try and keep the extracted contours a bit less erratic, a revised approach was taken. The average error over a larger region taken instead of using the pixel wise error. The averaging region was shrunk and the amplitude of the adjustment signal weakened for each iteration. The effect of this is that small local errors are not corrected as intensively as larger structural errors.
This algorithm was inspired by a technique used in video games for realistic procedural terrain generation, where layers of spatially coherent noise of increasing frequency and decreasing amplitude are summed. The technique produces natural looking structures and is used in games such as Minecraft [52].

4.1.4 Soft Actor Critic Auto Encoder

Soft Actor Critic Auto Encoder was employed to try and learn the optimal update function. Learning from a large state space is made possible by the latent representation produced by the encoder. However, acting on a large continuous action space is not solved that easily. Instead, a smaller action space was defined. The chosen action space is six-dimensional and maps to a bivariate gaussian. The adjustment DEM can be obtained from the gaussian and the update performed. Although this made the application of SAC-AE possible, it also made single updates very inefficient. Each iteration only has a local effect which means that subsequent states are very similar. This is bad for exploration and learning in addition to making the algorithm slow as many iterations have to be performed.

This problem can be reduced during inference by doing more actions on a single error tile by iteratively performing multiple actions on a single state by nulling the error close to the action center for each iteration. This effectively makes the algorithm able to work on larger portions of the DEM in each iteration and reduces the number of expensive environment interactions.

The initialization function for the SAC-AE algorithm was a bit different to the naive approach. First was the true DEM normal field smoothed, and then was normalization performed. The normalization consisted of subtracting the mean of the true DEM, rounded to the closest multiple of the contour interval, from the smoothed DEM and then dividing the smoothed DEM by the contour interval. The normalization was done to remove as much unnecessary information that varies between samples from the task as possible. This was an effort to make the algorithm more robust and learn quicker.

The algorithm was given the interpolation error of the normalized state, instead of working on the normalized state DEM directly. This was done to promote actions

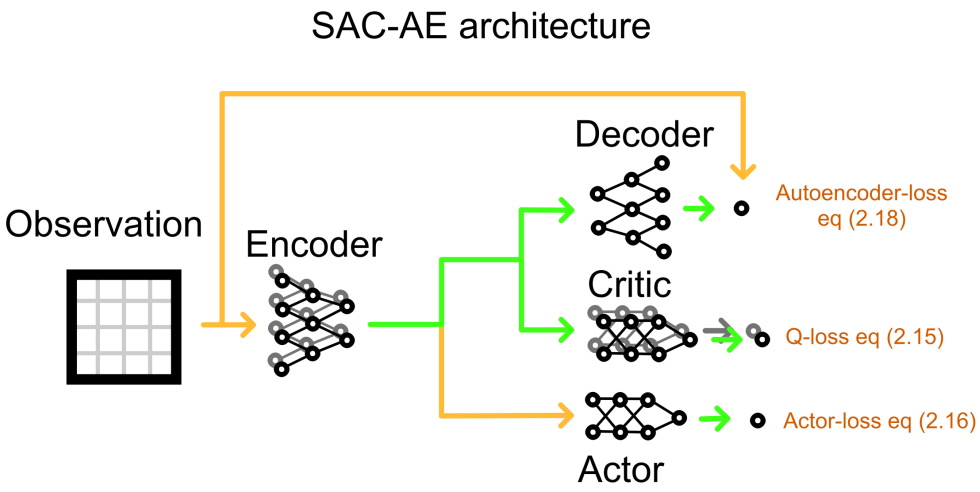


Figure 4.3: The soft actor-critic architecture. The yellow arrows propagate the signal forwards, but not derivatives backwards. The green arrows propagate both the signal forwards and the derivatives backwards. For stability is the critic represented both by a "live" version (in black) and "target" version which is as a sliding average of the previous "live" versions (in gray).

that have an effect on the interpolated DEM, as all actions affect the state DEM, but need not affect the contours extracted from the state and therefore not the interpolation.
The reward signal in each step was given by the negative scoring function $-S_\lambda(A|T)$. It was negated simply because SAC-AE tries to maximize the reward while $S_\lambda(A|T)$ scores good solutions low and bad solutions high.

4.1.5 Implementation

The authors of the SAC-AE paper implemented the algorithm using the PyTorch deep learning framework [53]. This implementation was used as a reference in my implementation of the algorithm using the Burn framework.
The encoder of the SAC-AE algorithm was implemented as three convolutional down-blocks followed by fully connected layer and a layer norm. Each convolutional block consisted of two convolutional layers; the first one with a stride of 2 to reduce the dimensionality of the input and the second with the standard stride of 1 and both with a padding of 1. Each convolutional block reduces the dimensionality of the data and after all three blocks the input of 256 by 256 pixels is flattened to a vector of 1024 elements. A fully connected layer maps the 1024 elements down to a vector in the latent space.
The decoder is a mirror of the encoder with a fully connected layer mapping the latent vector to a 1024 dimensional vector followed by three convolutional up-blocks. Each up-block consisted of a convolutional layer with padding and a convolutional transpose layer with stride 2 which upscales the data by a factor of two.

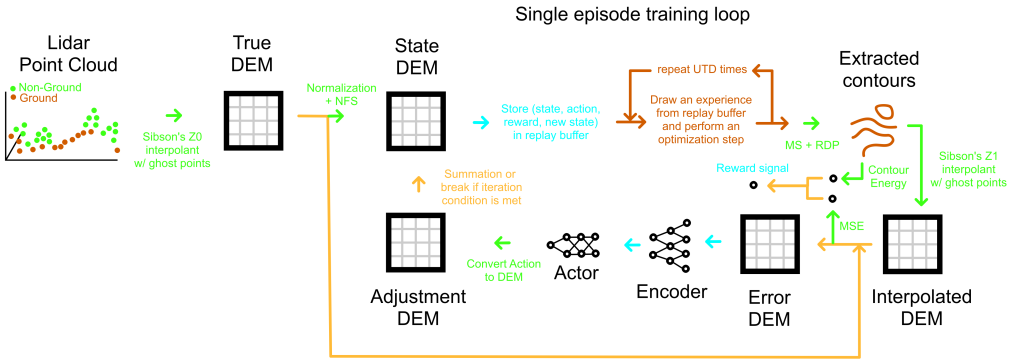


Figure 4.4: SAC-AE training loop for a single episode. The training has a warm up period until the replay buffer is filled up with experiences. During warm up is the brown parts of the loop skipped. Green represent fixed functions, yellow simple signal propagation and and blue and brown are SAC-AE elements.

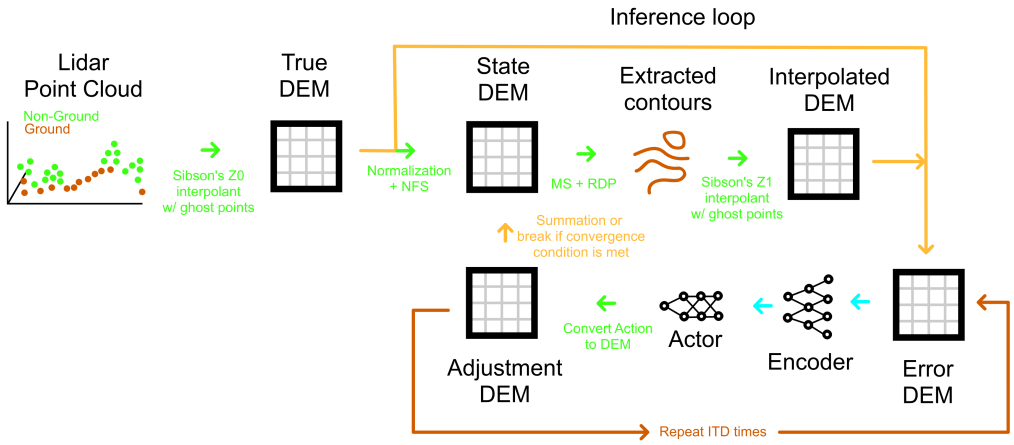


Figure 4.5: The inference loop for SAC-AE is similar to the training loop stripped from critic and decoder associated elements. The brown arrow is an optimization to limit the amount of times the expensive interpolation function is called. The chosen action is applied to the error DEM and a new action is chosen based on the new error. This is done ITD times before all actions are applied to the state DEM and a new true error is calculated. The loop breaks if the chosen action is of too small amplitude.

The resulting output of the decoder has the same shape as the input to the encoder. ReLU was used as the activation function for every layer in the autoencoder.
The actor was implemented as a multi-layer perceptron (MLP) with four layers. The input into the actor is the output of the encoder. The hidden layers had a size of 128 neurons and the output has dimensionality 12, 6 outputs for the mean value of the action distributions and 6 outputs for the logarithm of the standard deviations for each distribution. ReLU was again used as the activation function.
The critic consists of two Q-functions implemented as 3 layer MLPs with dropout. The dropout was added following the idea of the DroQ-paper for more efficient SAC learning and the dropout-rate was set to 50% [39]. The Q-functions used a hidden size of 64 neurons and output a single number corresponding to the estimated Q-value for the current state.
The entropy temperature was also learned and initialized with a value of 0.01.
Every module was optimized using the ADAM optimizer.

4.2 Supervised U-net

Supervised learning learns functions from a loss signal on a input-output pair. We want to learn a function that maps $\mathbb{R}^{M\times N}$ on $\mathbb{R}^{M\times N}$ without knowing any output examples. However, we have a scoring function that maps $\mathbb{R}^{M\times N}$ on $\mathbb{R}$. The function can unfortunately not be used as a loss signal, as the loss function must be differentiable and the scoring function $S_\lambda(A|T)$ is not. However, neural networks are universal approximation machines that can approximate any continuous function given enough neurons and data [33].
Let us assume that $S_\lambda(A|T)$ is continuous and approximate it with an ANN as ANNs are differentiable!

First was the evaluation network trained, see figure 4.6. The parameters of the evaluation was then frozen and the network was used as a loss function for training a U-net on the task of optimizing equation (4.1).
Generating the optimal DEM in one shot makes it difficult to ensure continuity across tile borders. One possible approach is to have a large overlap and apply a smoothing function between neighboring tiles.

4.2.1 Implementation

The evaluation network was implemented as a quite standard convolutional network similar to the encoder in the SAC-AE followed by two fully connected layers. As the input the evaluation network takes a state DEM and the true DEM stacked as two channels in a tensor. The convolutional part of the network consists of a convolutional layer with 8 3×3 filters with padding followed by four down blocks, each consisting of 2×2 maxpooling with a stride of 2 and two convolutional layers with 3×3 filters, reducing the dimensionality of the input to 32×32. This is flattened and fed into a fully connected layer reducing the dimensionality to 256. Dropout with a probability of 0.5 is performed on this layer. Before the final fully connected layer maps the latent vector to two elements, corresponding to the error

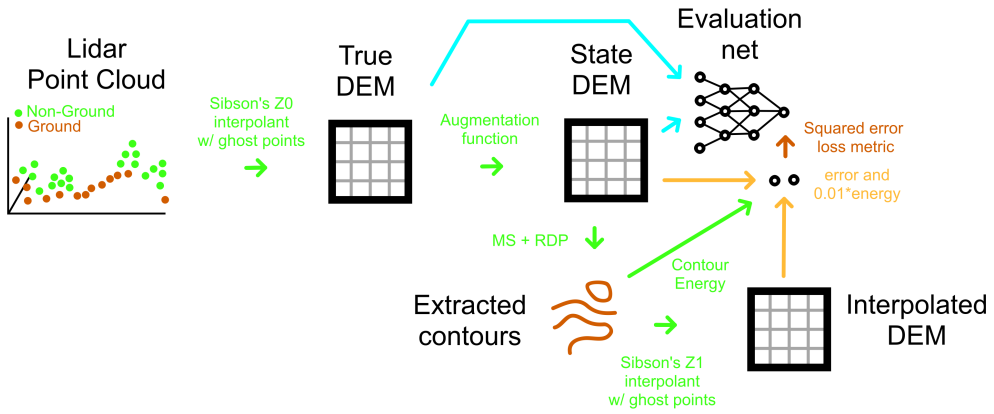


Figure 4.6: The training structure for the evaluation network. The network is trained in a supervised learning fashion to approximate $S_\lambda(A|T)$.

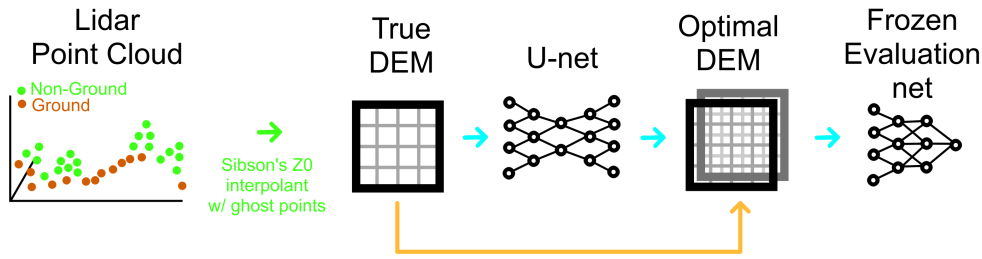


Figure 4.7: The U-net trained by a frozen evaluation network which in turn was trained to approximate $S_\lambda(A|T)$ with a differentiable function.

and (scaled) energy of the input. As in the earlier models was ReLU used as the activation function throughout the network and Adam as the optimizer.

The U-net implementation was similar to the original published network of Ronneberger et al. [36], only one layer shallower. The network was designed shallower to limit the number of parameters. The plan was to expand the network if the training of the shallow network proved successful.
The implementation was again similar to the encoder and decoder of the SAC-AE implementation. The network consists of an initial convolution block with 16 3×3 filters with padding followed by three down blocks and three up blocks and a final convolutional layer with a single 1×1 filter. Skip connections are made between the corresponding down and up blocks stacking the data in the channels dimension.
Each down block consists of 2×2 max pool with a stride of 2 followed by two convolutional layers with 3×3 filters with padding. The down blocks reduced the spatial dimensions of the input while increasing the channel dimension.
Each up block performed 2×2 cubic interpolation of the input to double the size of the input and was followed by two convolutional layers with 3×3 filters with padding. An up block doubles the spatial dimension of the input while reducing the channel dimension.
The final 1×1 convolution layer merge all the channels to a single channel tensor representing a DEM.
ReLU and Adam were as always used as the activation function and optimizer.

5
Results

The results will be twofold. The first subsection will discuss the contour scoring function S_λ, as defined in (3.1) with $\lambda = 0.01$, and present and discuss the results of the optimization algorithms. The second section will look at the application that was developed, named OmapMaker.

5.1 Contours

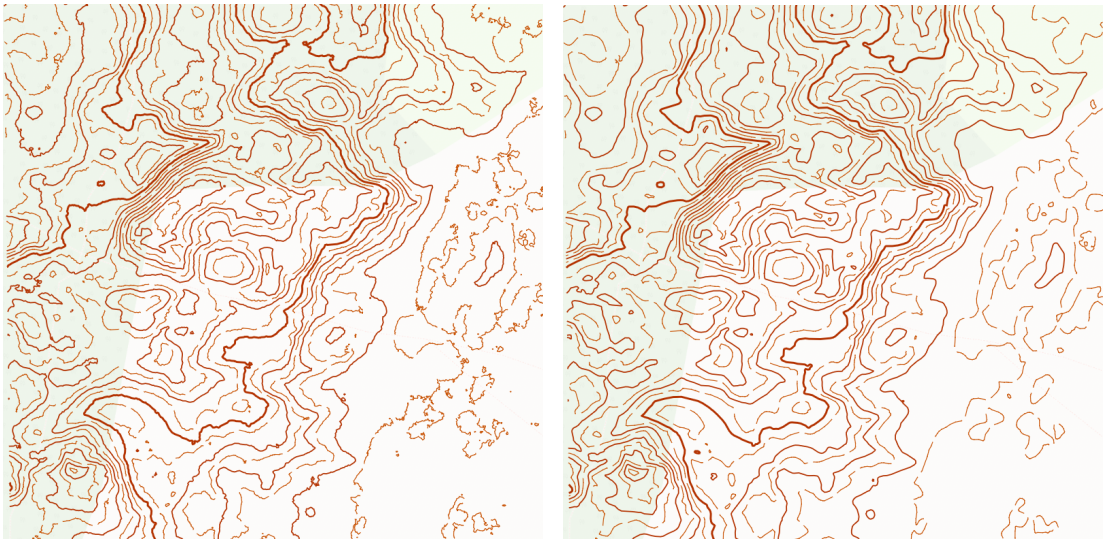
5.1.1 Scoring function

By visually comparing the scores given by the scoring function S and their corresponding contour sets, it seems that the scoring function reflects the subjective ranking of the quality of different contour sets over an area. Contours of higher degree of smoothness consistently achieve better scores than raw contours, see table 5.1. Erratic contours that are hard to read receive bad scores, even though the interpolation error might be low.

Terrain type	Raw Score	Smooth Score
Volcanic flats I	11.80	4.11
Volcanic flats II	8.68	4.58
Volcanic hill	6.39	3.67
Nordic moraine	10.21	5.42
Nordic coastal	10.40	5.02
Sand forest	23.83	6.40

Table 5.1: Contour scores for different raw vs smoothed contour sets for different terrain types. Smoothed contours consistently score better even though the interpolation error often worsened slightly. This is due to the punishing of squiggly contours. For visual comparison see figures 5.3 to 5.7.

Taking a look at figure 5.1 it is apparent that the information conveyed by the raw and the smoothed contour sets is very similar, but the erratic nature of the raw contours makes them harder to read. This is especially visible in the flatter regions of the maps. The contour scoring function agrees with this observation, scoring the smoother contours better even though the interpolation error actually increases from 3.71 to 5.24.
The scoring function could, unfortunately, not be applied to hand-made contours as the function depends on the winding order of the contours, which hand-made



(a) Raw contours achieve a score of 23.8.

(b) Smooth contours achieve a score of 6.4

Figure 5.1: The scoring function scores the smoother contour set lower, i.e. better, than the raw jagged contours, even though the raw contours achieve a lower interpolation error than the smooth contours. This is in line with how the author would rank the two contour sets. The example is from a sand-forest in southern Sweden and a contour interval of 2.5 meters with form lines is used. No post-processing of the contours are done before visualization.

contours do not respect. The same is true for the current state-of-the-art method, Karttapullautin.

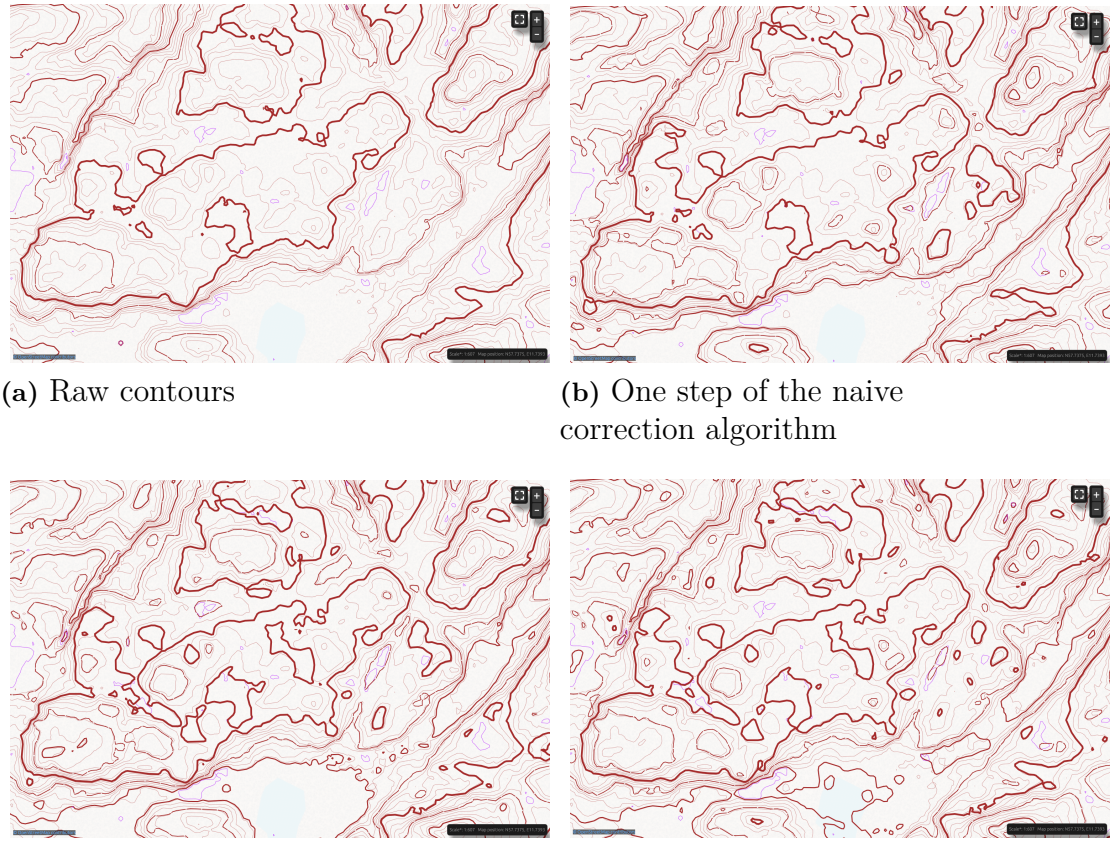
5.1.2 Optimization

The methods compared are as follows:

1. Raw contours
2. Normal field smoothed contours
3. Naive iterated contours with averaging.

SAC-AE and U-Net methods, although promising, have not yielded significant results due to time constraints. Their evaluation is left for future work.
Raw contours are simply the output from the marching squares algorithm on the true DEM, i.e the isolines of the landform at a selected contour interval. The resulting contours tend to be overly squiggly and hard to read, and do not try and adapt to the regions in between the contours whatsoever. The scores of raw contour sets are dominated by the high squigglyness and are therefore often quite high.

The normal field smoothed contours perform the smoothing on the DEM-level so the DEM values that are not exactly on the isolines also affect the results, but not in a way that aims to describe the terrain better. This often leads to slightly worse interpolation error, but the contour smoothness is massively improved leading to



(a) Raw contours

(b) One step of the naive correction algorithm

(c) Two steps of the naive correction algorithm

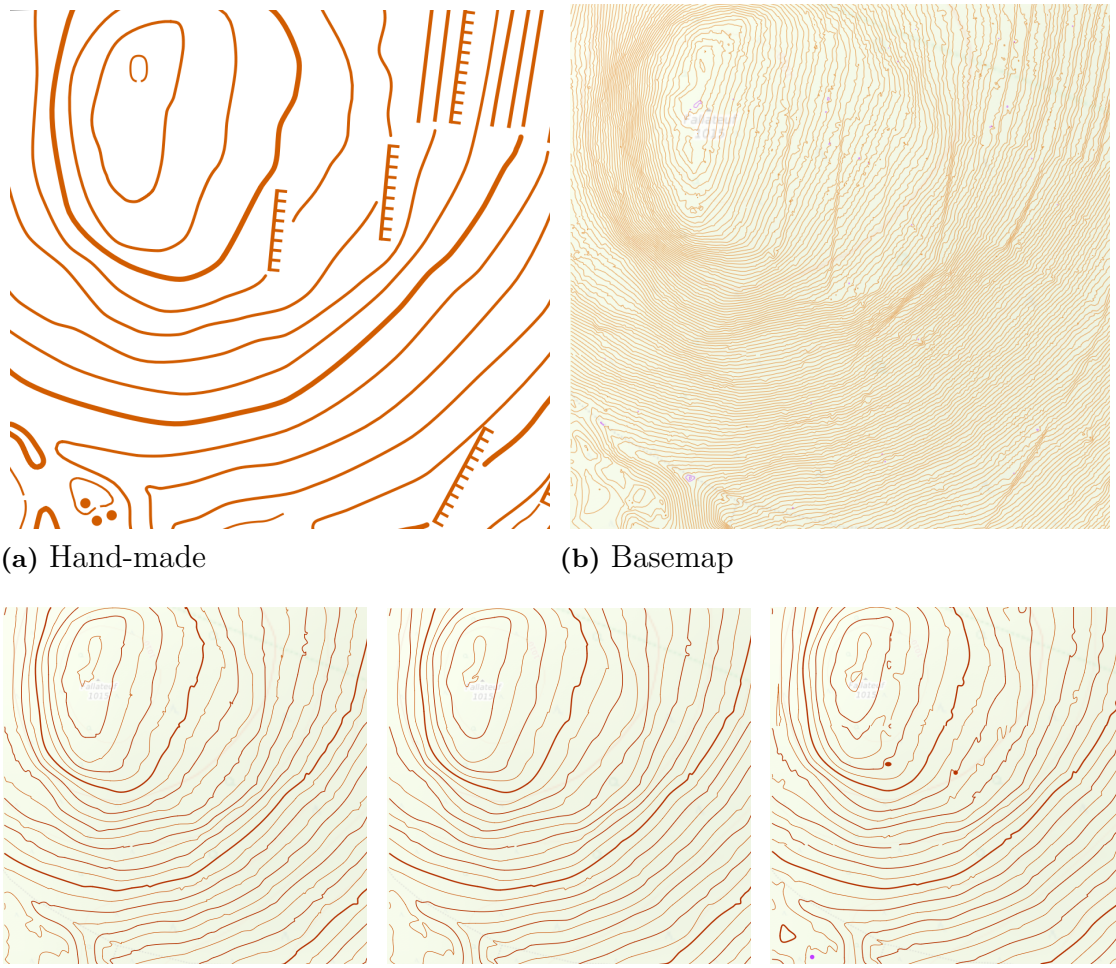
(d) Three steps of the naive correction algorithm

Figure 5.2: Visual comparison of the results of the naive interpolation algorithm. One step of the algorithm clearly marks knolls ignored by the raw contours. Increasing the step count leads to more erratic contours. This is expected as the algorithm does not take the contour energy into account.

better scores overall, compared to the raw contours.

The naive approach tries to iteratively adapt the DEM by elevating areas that get interpolated to low or vice versa. The main weakness of this approach is that it does not take into account the squigglyness of the contours. Each iteration step tend to add small contours cluttering the map. This can be seen in figure 5.2. Another weakness is that the method does not take method of how the contours are extracted and the interpolation method in to account. The approach just naively assumes that adding to the DEM should produce a corresponding addition to the interpolated DEM. This means that convergence is not certain.

A big flaw in the implementation is that the naive contours are not continuous across tile borders. An algorithm to ensure continuity was discussed in the Optimization section, but was not implemented due to time constraints.
In the following figures are the three methods of contour generation compared to the basemap and hand-made contours. Every sample uses a contour interval of 5 meters



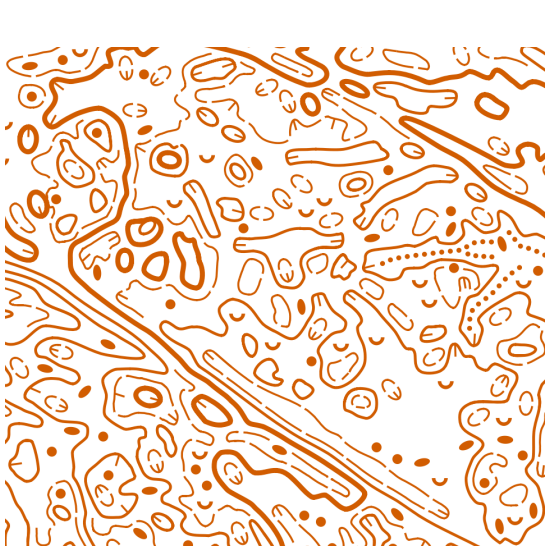
(a) Hand-made **(b)** Basemap

(c) Raw. Score: 6.39 **(d)** Smooth. Score: 3.67 **(e)** Naive. Score: 3.35

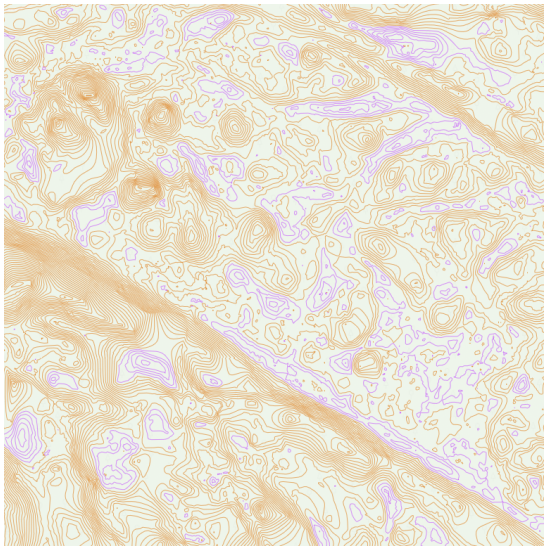
Figure 5.3: As expected do the generated contours quite will on smooth steep hills as steep hills means that the contours are close together. The earth banks from the hand-made map are quite prominent in the basemap, but neither of the methods do a good job showing them by bending the contours upwards. The smooth contours exaggerate the edge where the left most earth bank is located the best.

with form lines, effectively reducing the interval to 2.5 meters. Post-processing are used on all generated contours with both bezier fitting with a maximum error of 1 meter and marking contour loops with an area of less than 40 square meters as dot knolls or depressions. If the aspect of the dot knoll is more than 1.5 then an elongated dot knoll is added, see section A.2 in the appendix for more details on the post-processing. Purple denote depressions, this was done for simplicity of implementation, the real ISOM symbols are used in the final map files. The slight background color in the basemaps and generated contours are a result of the contour maps being overlaid OpenTopoMap in the developed application.

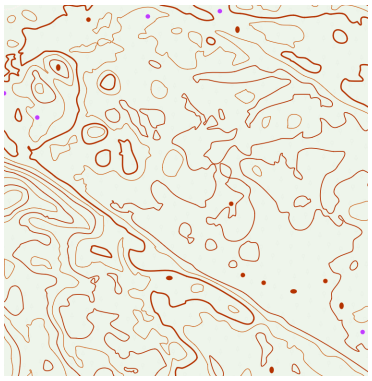
In summary, the smooth contours score better on average than the raw and the naive contours. The naive approach is the only method to use the interpolation error in its definition and none of the methods use the entire scoring function, which would be



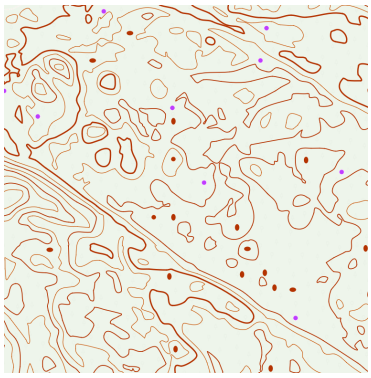
(a) Hand-made



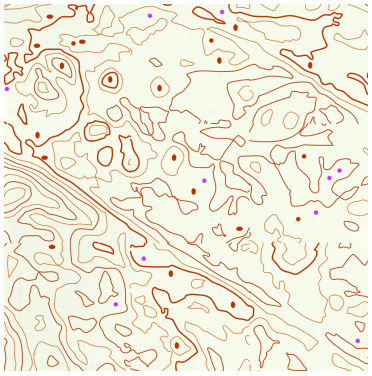
(b) Basemap



(c) Raw. Score: 11.80



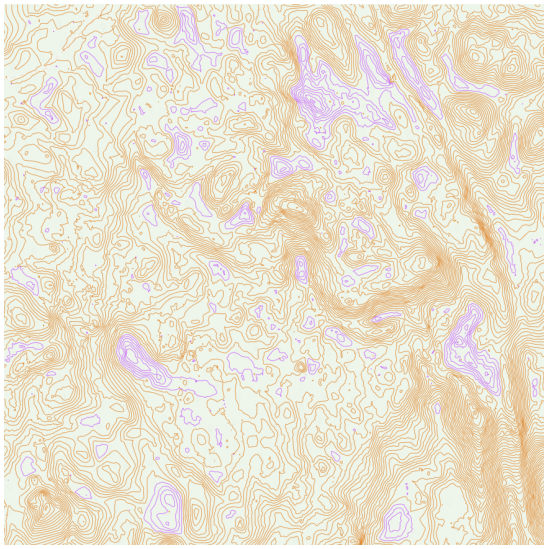
(d) Smooth. Score: 4.11



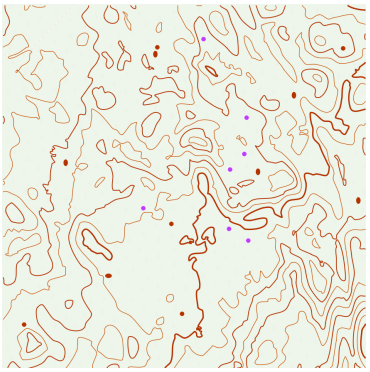
(e) Naive. Score: 5.47

Figure 5.4: Complex volcanic terrain without large changes in elevation are notoriously difficult to depict well by generated contours. Subjectively I would say that the smooth contours are the most similar to the hand-made contours. These also receive the best score.

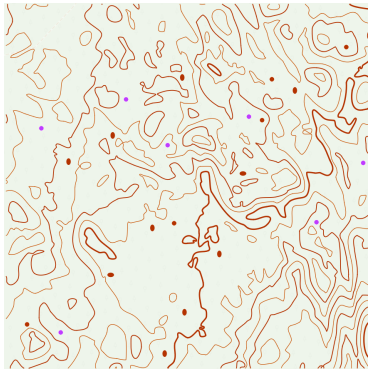
(a) Hand-made



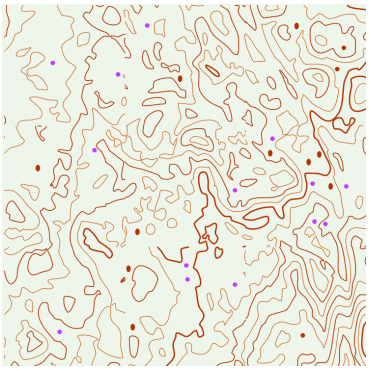
(b) Basemap



(c) Raw. Score: 8.68

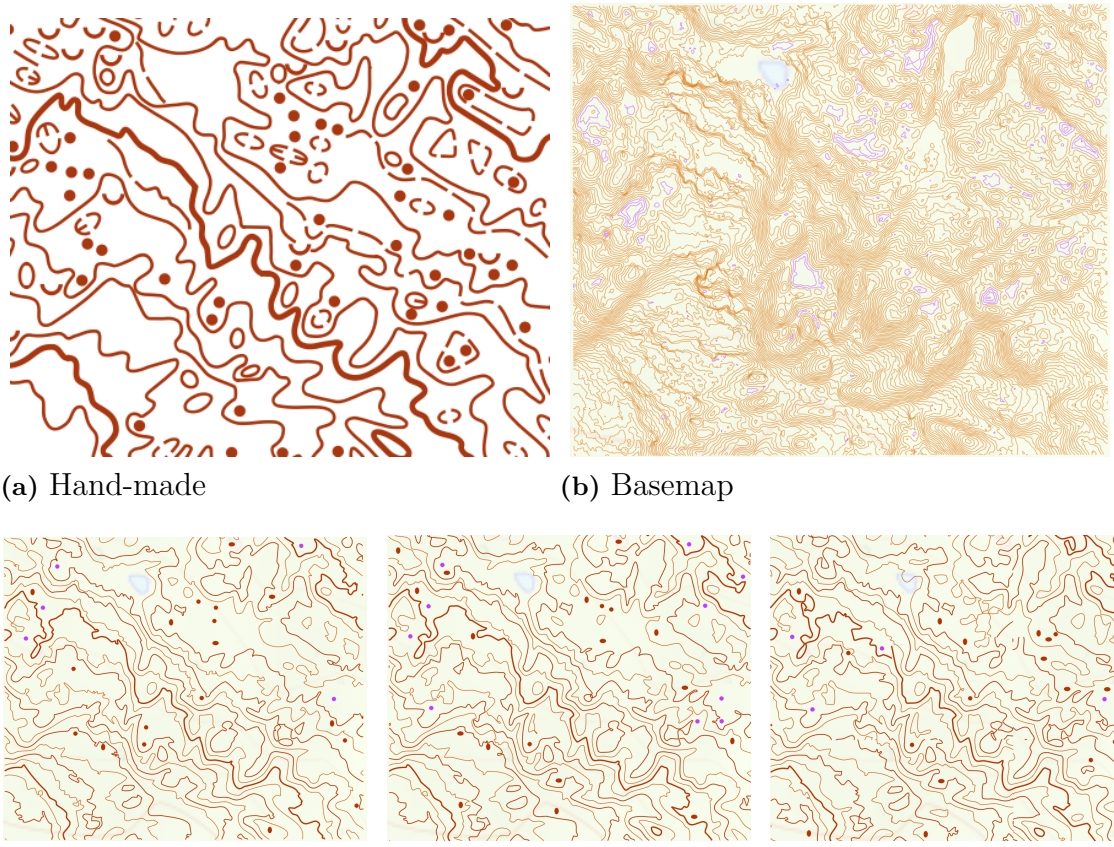


(d) Smooth. Score: 4.58



(e) Naive. Score: 4.34

Figure 5.5: Another sample from the same are as 5.4. Again I prefer the smooth contours over the raw, but interestingly it is the naive method that receives the best score. One can see that the naive approach have more small knolls in the flatter region and are more similar to the hand-made contours there. Of course, the discontinuities make it hard to do a fair comparison.

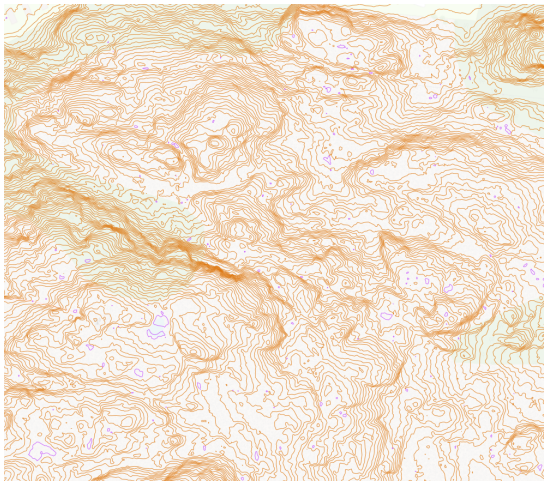


(a) Hand-made **(b)** Basemap

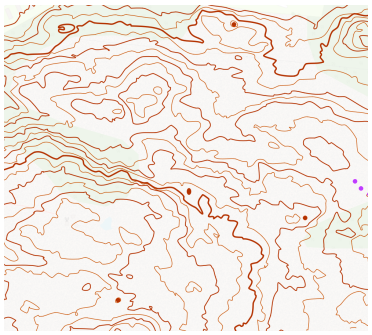
(c) Raw. Score: 10.21 **(d)** Smooth. Score: 5.42 **(e)** Naive. Score: 9.01

Figure 5.6: This map piece is from a Nordic hillside with many small details formed by large amounts of melting water scouring the hillside when the ice retreated over this part of Norway. Automatic generated contours traditionally struggle with such detailed areas. Again I would say that the smoothed contours are the best, but neither method do a good job in this complex terrain.

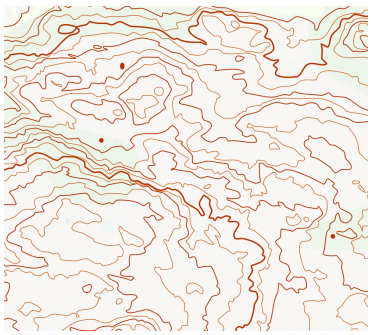
(a) Hand-made



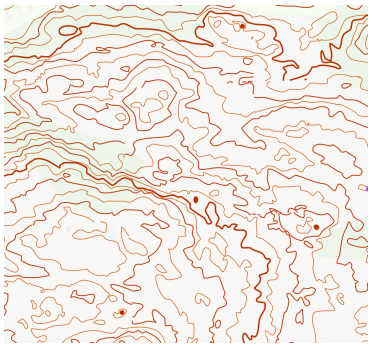
(b) Basemap



(c) Raw. Score: 10.40



(d) Smooth. Score: 5.02



(e) Naive. Score: 4.98

Figure 5.7: This map section is from a steep coastal terrain outside of Gothenburg. The hand-made contours are quite complex using many formlines, making it quite hard to read. As this terrain is both steep and with a lot of small details, are the generated contours expected to do better than the complex and flat areas, but worse than the smooth and steep area. In this area I would say that the naive method is the best as it picks up on more small details that the other two miss. The steepness also helps minimizing the discontinuities and contour energy.

necessary to significantly improve upon current methods, such as Karttapullautin.

5.2 OmapMaker

OmapMaker is the name of the application that was developed. The application guides the user step by step through the process of creating a map from lidar data. The application is fast and responsive running the frontend on its own thread and all the heavy calculations on multiple threads in the background. The front- and backend communicate through a message passing interface. The application is the first map application for orienteering maps with an interactive graphical user interface for adjusting parameters. OmapMaker writes orienteering map files with both point, line and polygon objects in vector format. The map files are georeferenced and magnetic north aligned given that the lidar files has a valid coordinate reference system (CRS) or the CRS is provided by the user, for more details on the georeferencing process see A.2 in the appendix. The GUI as a slippy map tile background map with both OpenStreetMap and OpenTopoMap as possible tile providers for easy geolocalization and overview. A polygon filter can be added by clicking out the polygon on the background map if only a part of the provided lidar files are to be mapped. The interactive parameter tuning and complete map file writing marks a significant improvement in the beginning of the map making workflow.

The application is designed to be easily extended with more functionality to make it a general purpose orienteering map generation program, and not only for contours. Figures 5.8, 5.9, and 5.10, show relevant screenshots of the application.

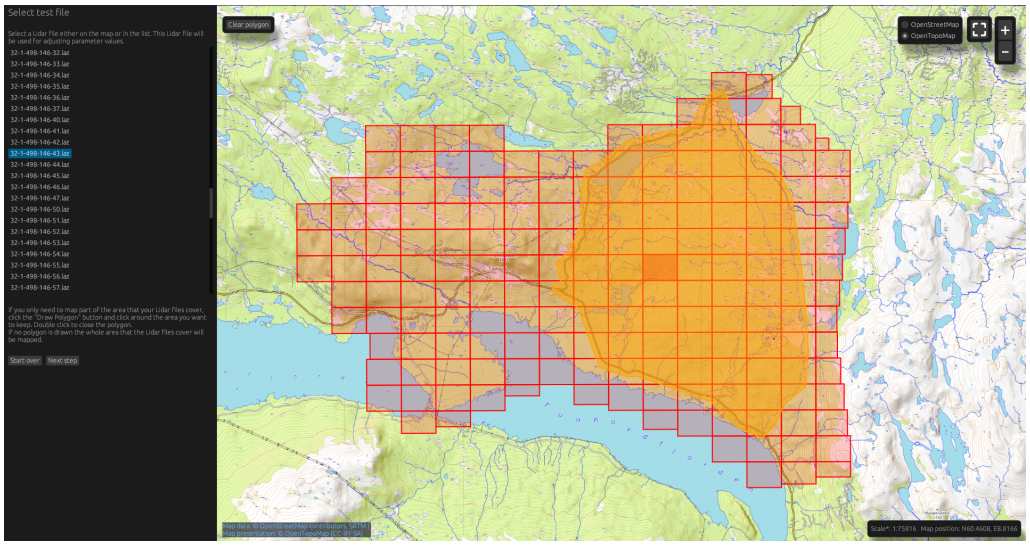


Figure 5.8: A polygon filter (orange) can easily be added to limit the area that has to be processed. The lidar file boundaries (red rectangles) and CRS are read from the header of the lidar files and automatically reprojected into the Web-Mercator projection and overlaid the background map. The lidar file to be used for parameter tuning can be chosen either by clicking the map or choosing it from the list.

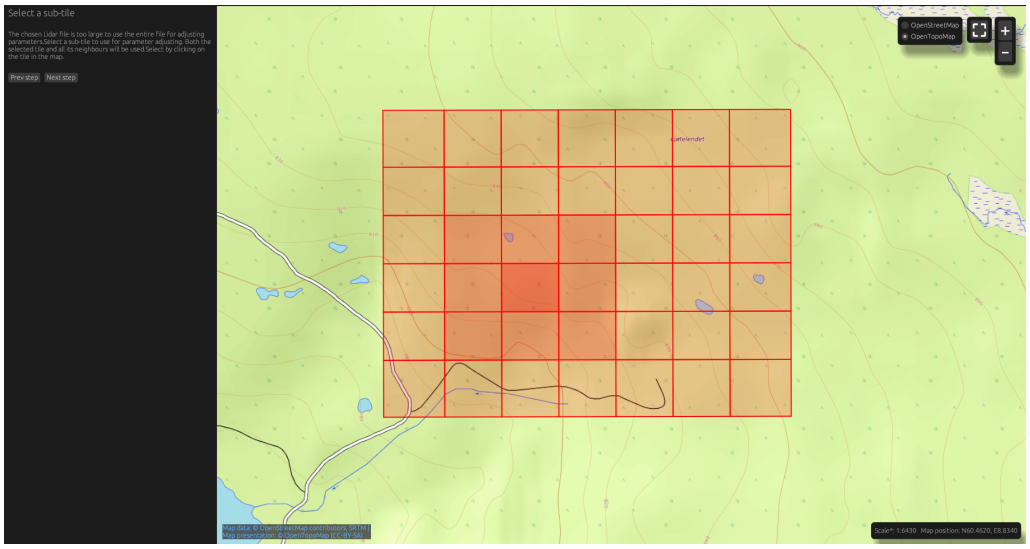


Figure 5.9: If the lidar file chosen for parameter tuning is too large, a sub-tile has to be chosen. This is to keep the iterations for parameter tuning quick. Both the chosen sub-tile and is neighbors are used for iterating on the map parameters. This runs on up to 9 threads in the background.

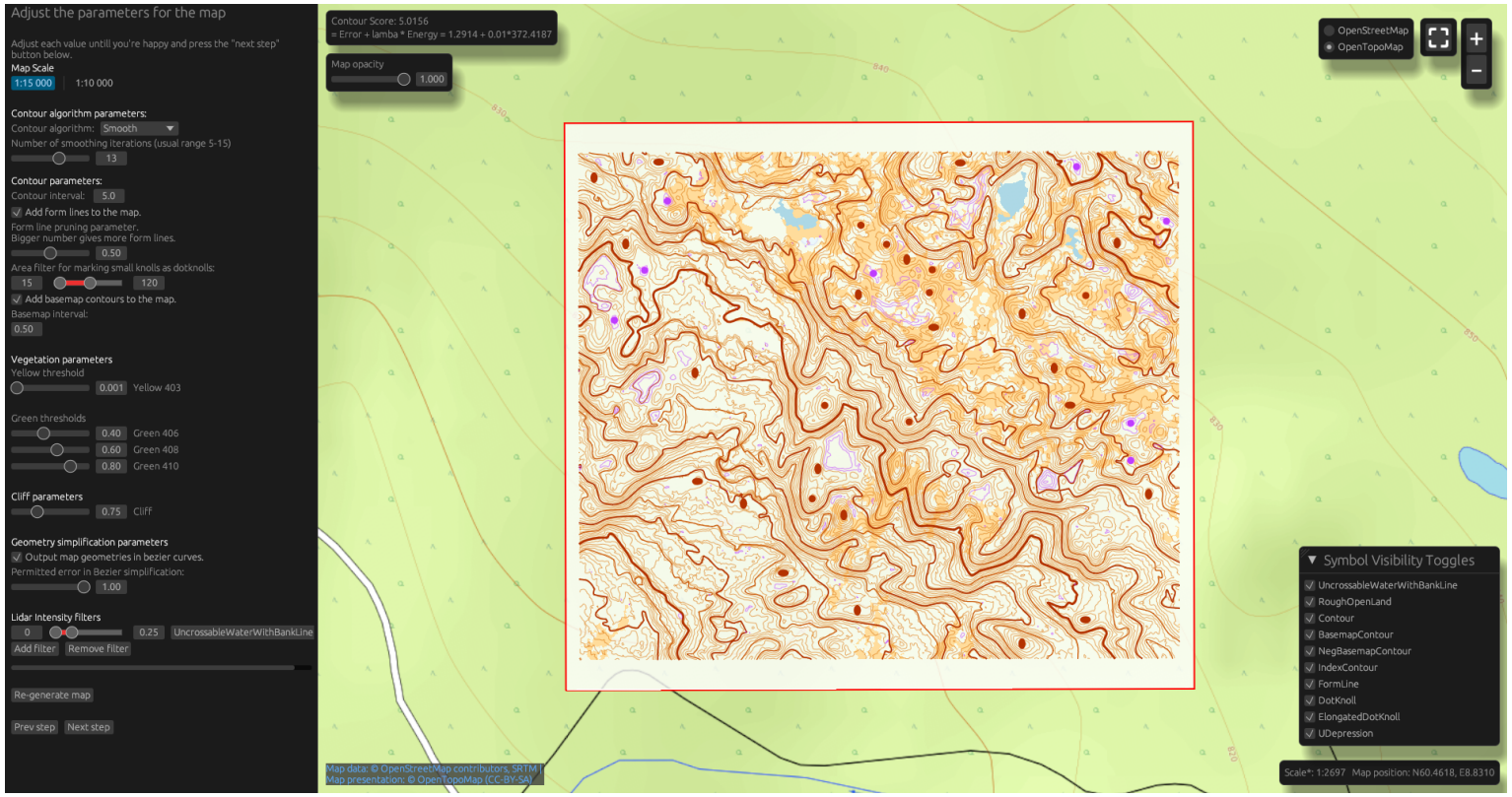


Figure 5.10: The parameter tuning is the most important part of the application. The contour interval can be adjusted and the contouring algorithm chosen. The number of iteration steps are tunable for the algorithms with that parameter. A base map with a high-resolution contour interval can be enabled to better show the true elevation profile. Both post-processing steps of the contour output from the chosen algorithm can be controlled. A dot knoll and depression filter removes the smallest closed contour loops and marks the second smallest as either dot-knolls or depressions. Smooth bezier curves can be fitted to the contours with a tunable maximum error tolerance. A new map image is generated based on the chosen parameters. Even simple vegetation filters and a lidar intensity filter mapped to a water symbol are enabled in the figure. The contour score is visible in the top left corner of the map panel.

6
Conclusion

This thesis aimed to define an objective measure for the quality of a contour set taking into account how experienced map readers interpret contours, as well as investigate methods for optimizing on a contour set. In the process, an application for displaying contour maps was developed. The application facilitates easy comparisons between contouring algorithms and makes the tuning of algorithm parameters efficient.

The measure for defining the quality of a contour set was successfully defined and consistently gave better scores to visually smooth contours that preserved the information from the raw contours. Optimizing the contour sets using the quality measure proved difficult. This was partially due to the non-differentiability of the scoring function, but also because the function being expensive to compute. Both naive methods as well as machine learning were employed in the quest to optimize the contours.

The naive methods delivered unsatisfactory result due to the absence of an energy term in the optimization pipeline, but also showed promising signs for using an interpolation based error metric in the evaluation of a contour set represented by DEMs.

Two machine learning methods were proposed and developed, but neither were trained successfully.
The SAC-AE approach phrases the optimization problem as a reinforcement learning problem with an environment in which an agent takes actions and learns based on the received rewards. While flexible and maybe more human-like in it's actions, is SAC-AE quite complex and expensive to both train and use for inference as each action has only a local effect which makes subsequent states quite similar.
The U-net approach, traditionally used in supervised learning, has the problem of having no training data available. To circumvent this, a second network was trained to approximate the scoring function, with the advantage that the neural network is differentiable. The differentiable evaluation network was then to be used as the loss function for the U-net. This approach has the advantages of being able to handle larger DEM tiles as input compared to SAC-AE and is more efficient in inference.
Their full implementation and successful training are left for future work. All code produced when working on this project is available on the authors Github[1] or is

[1] https://github.com/yvind

merged into to open-source libraries.

The developed application makes parameter tuning easy and iterations quick. It makes comparing contour algorithms easy and the best algorithm for a test area can quickly be determined before generating the entire map. The written map files are georeferenced oriented to the magnetic north.
The application represents a significant improvement to the beginning of the map-making process enabling better control over the output and speeding up the process of going from lidar data to a vector map.

Bibliography

[1] F. Bandini, T. P. Sunding, J. Linde, O. Smith, I. K. Jensen, C. J. Köppl, M. Butts, and P. Bauer-Gottwein, "Unmanned aerial system (uas) observations of water surface elevation in a small stream: Comparison of radar altimetry, lidar and photogrammetry techniques," *Remote Sensing of Environment*, vol. 237, p. 111487, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0034425719305061

[2] B. Lohani and S. Ghosh, "Airborne lidar technology: A review of data collection and processing systems," *Proceedings of the National Academy of Sciences, India Section A: Physical Sciences*, vol. 87, 11 2017.

[3] P. Kettunen, C. Koski, and J. Oksanen, "A design of contour generation for topographic maps with adaptive dem smoothing," *International Journal of Cartography*, vol. 3, no. 1, pp. 19–30, 2017.

[4] P. Raposo, "Variable dem generalization using local entropy for terrain representation through scale," *International Journal of Cartography*, vol. 6, no. 1, pp. 99–120, 2020.

[5] M. Hofer, G. Sapiro, and J. Wallner, "Fair polyline networks for constrained smoothing of digital terrain elevation data," *IEEE Transactions On Geoscience And Remote Sensing*, 2006. [Online]. Available: https://www.geometrie.tugraz.at/wallner/dted.pdf

[6] J. B. Lindsay, A. Francioni, and J. M. H. Cockburn, "Lidar dem smoothing and the preservation of drainage features," *Remote Sensing*, vol. 11, no. 16, 2019. [Online]. Available: https://www.mdpi.com/2072-4292/11/16/1926

[7] B. Jenny, "Terrain generalization with line integral convolution," *Cartography and Geographic Information Science*, vol. 48, no. 1, pp. 78–92, 2021.

[8] X. Sun, P. L. Rosin, R. Martin, and F. Langbein, "Fast and effective feature-preserving mesh denoising," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 5, pp. 925–938, 2007.

[9] G. B. Folland, *Fourier Analysis and Its Applications*. American Matheatical Society, 2009.

[10] G. Farin, J. Hoschek, and M.-S. Kim, *Handbook of computer aided geometric design*. Elsevier Science B.V., 2002.

[11] M. Kass, A. P. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision*, vol. 1, pp. 321–331, 2004. [Online]. Available: https://api.semanticscholar.org/CorpusID:12849354

[12] T. Bayer, I. Kolingerová, M. Čelonk, and J. Lysák, "Simplification of contour lines, based on axial splines, with high-quality results," *International Journal of Geographical Information Science*, vol. 37, no. 7, pp. 1520–1554, 2023.

[13] J. Ryyppö. [Online]. Available: https://www.routegadget.net/karttapullautin/
[14] D.-J. Kroon, "Isocontour, matlab central file exchange," 2025. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/30525-isocontour
[15] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *Siggraph Comput. Graph.*, vol. 21, no. 4, p. 163–169, Aug. 1987. [Online]. Available: https://doi.org/10.1145/37402.37422
[16] U. Ramer, "An iterative procedure for the polygonal approximation of plane curves," *Computer Graphics and Image Processing*, vol. 1, no. 3, pp. 244–256, 1972. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0146664X72800170
[17] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica*, vol. 10, no. 2, pp. 112–122, 1973.
[18] W. Shi and C. Cheung, "Performance evaluation of line simplification algorithms for vector generalization," *The Cartographic Journal*, vol. 43, no. 1, pp. 27–44, 2006.
[19] R. Ziatdinov, "Bernstein polynomials: a bibliometric data analysis since the year 1949 based on the scopus database," 2025. [Online]. Available: https://arxiv.org/abs/2503.07614
[20] T. Bobach, "Natural neighbor interpolation - critical assessment and new contributions," Ph.D. dissertation, Technical University of Kaiserslautern, 2008.
[21] J. Flötotto, "A Coordinate System associated to a Point Cloud issued from a Manifold: Definition, Properties and Applications," Theses, Université Nice Sophia Antipolis, Sep. 2003. [Online]. Available: https://theses.hal.science/tel-00832487
[22] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data," in *Proceedings of the 1968 23rd ACM National Conference*, ser. ACM '68. New York, NY, USA: Association for Computing Machinery, 1968, p. 517–524. [Online]. Available: https://doi.org/10.1145/800186.810616
[23] S. Y. Chung, S. Venkatramanan, H. E. Elzain, S. Selvam, and M. Prasanna, "Chapter 4 - supplement of missing data in groundwater-level variations of peak type using geostatistical methods," in *GIS and Geostatistical Techniques for Groundwater Science*, S. Venkatramanan, M. V. Prasanna, and S. Y. Chung, Eds. Elsevier, 2019, pp. 33–41. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128154137000043
[24] R. Sibson, "A brief description of natural neighbor interpolation," in *Interpreting Multivariate Data*, V. Barnett, Ed., 1981, pp. 21–36.
[25] ——, "A vector identity for the dirichlet tessellation," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 87, no. 1, pp. 151—-155, 1980.
[26] G. Farin, "Surfaces over dirichlet tessellations," *Computer Aided Geometric Design*, vol. 7, no. 1, pp. 281–292, 1990.
[27] D. VandenHeuvel. [Online]. Available: https://danielvandh.github.io/NaturalNeighbours.jl/dev/compare/#Quantitative-Local-Analysis
[28] T. Bobach, M. Bertram, and G. Umlauf, "Comparison of Voronoi based scatterd data interpolation schemes," in *Proceedings of the Internationl Conference on*

Visualization, Imaging and Image Processing, J. Villanueva, Ed., 2006, pp. 342–349.

[29] T. Bobach, G. Farin, D. Hansford, and G. Umlauf, "Natural neighbor extrapolation using ghost points," *Computer-Aided Design*, vol. 41, no. 5, pp. 350–365, 2009, voronoi Diagrams and their Applications. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S001044850800167X

[30] J. Wallner, "Note on curve and surface energies," *1*, 1. [Online]. Available: www.geometrie.tugraz.at/wallner/invdiff.pdf

[31] T. Miura, "Elastic curves and phase transitions," *Mathematische Annalen*, vol. 376, no. 3–4, p. 1629–1674, Mar. 2019. [Online]. Available: http://dx.doi.org/10.1007/s00208-019-01821-8

[32] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, 2015.

[33] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208

[34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986. [Online]. Available: https://api.semanticscholar.org/CorpusID:205001834

[35] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," 2015. [Online]. Available: https://arxiv.org/abs/1511.08458

[36] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015. [Online]. Available: https://arxiv.org/abs/1505.04597

[37] U. Michelucci, "An introduction to autoencoders," 2022. [Online]. Available: https://arxiv.org/abs/2201.03898

[38] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," 2019. [Online]. Available: https://arxiv.org/abs/1812.05905

[39] T. Hiraoka, T. Imagawa, T. Hashimoto, T. Onishi, and Y. Tsuruoka, "Dropout q-functions for doubly efficient reinforcement learning," 2022. [Online]. Available: https://arxiv.org/abs/2110.02034

[40] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus, "Improving sample efficiency in model-free reinforcement learning from images," 2020. [Online]. Available: https://arxiv.org/abs/1910.01741

[41] *SAS/STAT® 14.1 User's Guide The Tpspline Procedure*, SAS Institute Inc., 2015. [Online]. Available: https://support.sas.com/documentation/onlinedoc/stat/141/tpspline.pdf

[42] Rust-lang. [Online]. Available: https://www.rust-lang.org/

[43] Burn. [Online]. Available: https://burn.dev/

[44] E. Ernerfeldt. [Online]. Available: https://crates.io/crates/eframe

[45] GeoRust. [Online]. Available: https://georust.org/

[46] S. Altmayer. [Online]. Available: https://crates.io/crates/spade

[47] C. E. Shannon, "Communication in the presence of noise," in *Proceedings of the I.R.E*, 1 1949.

[48] *LAS Specification 1.4 - R15*, The American Society for Photogrammetry & Remote Sensing, 2019.
[49] *Cloud Optimized Point Cloud specification 1.0*, Hobu Inc. [Online]. Available: https://copc.io/
[50] *LAZ Specification 1.4 - R0*, Rapidlasso GmbH, 2014.
[51] W. Schroeder, R. Maynard, and B. Geveci, "Flying edges: A high-performance scalable isocontouring algorithm," in *2015 IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV)*, 2015, pp. 33–40.
[52] T. Rose and A. G. Bakaoukas, "Algorithms and approaches for procedural terrain generation - a brief review of current techniques," in *2016 8th International Conference on Games and Virtual Worlds for Serious Applications (VS-Games)*, 2016.
[53] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus, "Pytorch soft actor critic autoencoder." [Online]. Available: https://github.com/denisyarats/pytorch_sac_ae

A

Appendix 1

A.1 Form lines

Orienteering maps allow the intermittent use of intermediate contours in regions where the normal contour interval is too coarse to depict all distinct landforms. These contours are called form lines and are depicted with dashed brown lines. Form lines should be used sparingly so as not to clutter the map. The contour interpolation function can be used to create a mask for where form lines are needed. The form lines algorithm starts by generating the contour lines at half the desired contour interval, to include all possible formlines, and measuring the pixel-wise interpolation error with respect to the true DEM everywhere. Then every form line is discarded and the interpolation error is calculated again. The worsening in interpolation error can thresholded to generate a pixel mask for where formlines are needed. The mask will benefit from some morphology operations, such as opening and closing, to remove the smallest mask areas and exaggerate other areas. Taking the intersection of the mask polygons and the form lines yields form lines in the areas where they reduce the interpolation error the most.

A.2 Writing map files

Post-Processing

The contours produced by the algorithms presented in chapter 4 produce strictly polyline contours. However, contours on an orienteering map can be represented by a wider range of symbols, including point symbols for small knolls and depressions. To make it easier for the user to read large elevation changes are every fifth contour a bit thicker and is called an index contour. This enables the user to do quick estimates of height differences over large areas. As discussed in the introduction 1.1, contours are modeling a smoothly changing phenomenon and should themselves be smooth. The output polyline contours are approximated by cubic bezier splines to make them smooth.

Dot knolls and depressions

The smallest contour features on orienteering maps are represented using point symbols to simplify the map reading. The post-processing in this work uses three distinct

point features to increase the readability of the map: dot knolls, elongated dot knolls and U-depressions.
The GUI exposes an area filter for which closed contour loops that should be converted to point symbols. The filter consists of a minimum and a maximum area. Small loops smaller than the minimum area is removed entirely from the map. Loops with areas within the filter range are converted to either dot knolls or U-depressions depending on wether the loops represents an increase or decrease in elevation. Elongated dot knolls are used for loops within the filter range where the aspect ratio of the minimal bounding box of the contour loop is greater than 1.5. The elongated dot knoll is rotated to have its long axis in the same direction to the longest side in the minimal bounding box. Every dot symbol is placed at the center of mass for the contour loops they replace.

Bezier

An efficient type of simplification is the conversion from polylines to bezier splines. This makes the assumption that the phenomena that the polyline represents are continuous in nature, which polylines (unless sampled infinitely dense) are not able to model. Bezier splines have the advantage of being able to model non-linear segments and smooth transitions between segments with relatively few vertices. Cubic bezier splines are frequently used in cartography, in font definitions and in illustration software, where it is better known as a path. Most algorithms that fit a bezier spline to a polyline only test for the bezier fit at the polyline vertices, but that can lead to large disparities in between the vertices where the polyline truly depict a corner. My implementation also tests the halfway point between vertices for the bezier fit to mitigate this. The algorithm described in the theory section is recursive and the base case of trying to fit a bezier to a single bezier segment is handled by adding the straight segment to the spline. This breaks the C^1 continuity of the spline at the straight segment's endpoints, but has the advantage of being able to model sharp corners. Beziers are harder to work with than polylines. The bezier simplification is therefore only applied after the final contour set is computed. The allowed error in bezier simplification is a parameter which can be tuned in the GUI of the application to easily be able to find a value that makes sense for a terrain.

Georeferencing, the three norths and two scale factors

Georeferencing is the process of assigning a real world geographic position to a map. This is done by identifying which projected coordinate reference system the lidar files are in and unprojecting a chosen map center from that projected CRS to a geographic CRS. The axes of an orienteering map are magnetic, meaning that the positive y-axis should be directed towards the magnetic north pole. However, both the projected CRS and the geographic CRS have their own norths.

True North (geodetic north/geographic north) is the direction towards the point where the Earth's axis of rotation intersects the surface of the globe in the northern hemisphere. Projected north (grid north) is the direction of the meridians on a flat

map projection. Magnetic north is the direction where a compass needle points. This direction changes over time as the magnetic north pole wanders. The Earth's magnetic pole has moved by between 55 and 35 kilometers a year this century (it is also magnetically a south pole, but that is irrelevant).

All three of the Norths are important when georeferencing a map. The lidar files are given in a projected coordinate reference system, where the y-axis points in the direction of the projected north. The angle between the true north and the projected north at a given location and projected coordinate system is called the convergence. The angle between the true north and the magnetic north is called the declination and is dependent on the location, date and elevation. Grivation is the difference between the convergence and the declination and is the angle between the projected north and magnetic north, which is used to rotate the projected coordinates to the map's magnetic coordinates.

When transforming distances in the projected Euclidean grid to real distances, two scale factors are needed. The first scale factor is called the grid scale factor transforms distances on the projected grid to distances on the mathematical ellipsoid that the geographic coordinate system defines. The second scale factor is the elevation scale factor that scales distances on the ellipsoid to distances on the ground and is simply defined as the ratio of the ellipsoid radius over the sum of the ellipsoid radius and the ground height relative to the ellipsoid. Both scale factors are functions of the geographic position on the globe and the grid scale factor is also a function of the projected coordinate reference system and is related to the convergence. The product of the two scale factors is used to transform projected distances to real distances and is most often very close to 1, often within 1 per mille.

Given the grivation, the combined scale factor and the maps scale, can coordinates in the projected CRS be transformed into the maps CRS. Omap files store map coordinates as an integer, giving the number of micrometers of paper the coordinate is away from the reference point with magnetic axis and the y axis flipped to match the y-axis of computer screens.

The transformation from a projected CRS to the map file coordinates is described by equation (A.1).

$$\boldsymbol{x}_{map} \approx \kappa\sigma\frac{1}{\chi}\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}\begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix}(\boldsymbol{x}_{crs} - \hat{\boldsymbol{x}}_{crs}) + \hat{\boldsymbol{x}}_{map} \tag{A.1}$$

where σ is the scale, f.ex 1/15000 or 1/10000, κ is a conversion factor from the units of $\boldsymbol{x}_{crs}$ into micrometers, f.ex 10^6 for meters to micrometers, χ is the combined scale factor and θ is the grivation and the hatted coordinate vectors are corresponding reference points in the projected CRS and map coordinates. The approximately equals sign signifies that the map coordinates are rounded to the closest integer.

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY