Eindhoven University of Technology

MASTER

Cartographically inspired visualization of points sets

van Arnhem, B.

Award date: 2011

Disclaimer
This document contains a student thesis (bachelor’s or master’s), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

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

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

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
Cartographically Inspired Visualization of Points Sets

Bart van Arnhem

April 10, 2011

Supervisor
prof.dr.ir. J.J. (Jack) van Wijk
Abstract

The display of a large number of points, distributed over the plane, is a classic visualization problem. Available space is usually limited, which means a simple plot of the points (known as a scatter plot) quickly becomes cluttered and also suffers from problems such as overplotting. An alternative solution is the density plot, which depicts point density, visualized using different color tones or elevation in a 3rd dimension.

We develop and evaluate a model that is heavily inspired by the standard topographic map, where not each individual is shown, but rather grouped, and shown as cities or villages. We take this metaphor, and apply it to visualize arbitrary sets of points in a plane. Our research is based on two initial observations on standard topographic maps: regions (such as cities, green-land, and industrial areas) are displayed as simple shapes, with uniform colors and crisp boundaries, sometimes labeled and accompanied by a symbol hinting its size, and regions are more or less uniformly distributed over the map according to their local importance, e.g., it is possible that a relatively small city in a sparse area is shown while a larger city in a denser area is not.

We consider scalability of the model, i.e., the ability to generate visualizations at different levels of detail, with smooth and predictable transitions (e.g., for interaction or animation purposes) to be an important factor. This, along with what we consider important properties of the map (such as region shape and size), is evaluated with a user study.
# Contents

1 Introduction .................................................. 2

2 Background .................................................. 4
   2.1 Visual representation ...................................... 5
   2.2 Scalability ................................................... 8

3 Problem Analysis ............................................. 10
   3.1 Alternative Choices ......................................... 11
   3.2 Input .......................................................... 13
   3.3 Region discovery ............................................ 14
   3.4 Visual representation ....................................... 15

4 Model .......................................................... 18
   4.1 Triangulation .................................................. 19
   4.2 Clustering .................................................... 19
   4.3 Density maps .................................................. 20
   4.4 Contour tree ................................................ 23
   4.5 Selecting regions ............................................ 28

5 Implementation ................................................ 34
   5.1 Input .......................................................... 34
   5.2 Density maps .................................................. 36
   5.3 Rendering Regions .......................................... 40
   5.4 Contour tree ................................................ 44
   5.5 Map layers .................................................... 47

6 Evaluation ..................................................... 50

7 Conclusions .................................................. 60

A Evaluation method ............................................ 66

B Batch export .................................................. 68

C User study data ............................................... 72

D Derivations .................................................... 80
Chapter 1

Introduction

The display of a large number of points, distributed over the plane, is a classic visualization problem. Available space is usually limited, which means a simple plot of the points, known as a scatter plot (Fig. 1.1(a)), quickly becomes cluttered and also suffers from problems such as overplotting. An alternative solution is the density plot, which depicts point density, visualized using different color tones (Fig. 1.1(b)) or elevation in a 3rd dimension (Fig. 1.1(c)). Well known examples are weather charts, showing temperature at different regions on a map.

Although not immediately obvious, solutions can also be found in a more ubiquitous type of map, the standard topographic map. As an example, we consider population visualized on such maps. Here not each individual is shown, but rather grouped, and shown as cities or villages. These visualizations are easy to interpret, familiar to the user and look attractive. We refer to this as the city metaphor, i.e., the use of crisp simple region shapes, with a focus on a more or less uniform distribution of the regions according to their local importance.

Suppose we are given a set of points, distributed over the plane with possibly a single attribute, and no ancillary information such as predefined clusters or boundaries. The central research question is now

*How to convert such a set of points into sets of crisp, simple regions that clearly depict the distribution of the points?*

This question leads to new questions, because for the actual visualization, there is probably not a single perfect solution. For example, the complexity of the shape of a region is a trade-off between simplicity and accuracy of the map. This leads to the following questions and challenges:

*What are relevant properties that characterize such regions, how can we generate regions according to different valuations for these properties, and what are the preferences of people for values?*
The aim of this thesis is to develop a model that can be used to automatically generate a variety of such regions, and to explore the different choices possible. Examples of real-world applications of such maps are for usage in atlases, or in general, to do spatial analysis on datasets containing large numbers of points.

**Thesis structure.** Chapter 2 compactly discusses the background of what we consider to be the higher goal of visualizing these points sets, which is data exploration. In Chapter 3 we look at the problem in more detail by asking ourselves questions such as *What shape do we expect the regions to have?* and *How do we distribute them over the available space?* Already proposing some algorithms and methods we roughly define the scope of the project and later on narrow this down and focus on what we think are the important aspects. Chapter 4 describes the different techniques, methods and data structures that form the important components of the model. Implementation specific details are described in Chapter 5. Evaluation of the model is done by a user study and presented in Chapter 6. We conclude this thesis with a short summary of our work (Chapter 7), and look ahead by discussing future work.
Chapter 2

Background

In this chapter we discuss the background to our problem statement. We approach this by looking at what we consider to be one of the primary functions of point set visualizations: data exploration. Within the context of data exploration we can broadly make a distinction between methods aimed at identifying patterns in a single, or between multiple attributes (univariate/multivariate analysis), change in patterns over time (temporal analysis), and patterns in the spatial distribution of the data (spatial analysis). Often these overlap, e.g., identifying changes in spatial patterns over time (spatial-temporal) or relations between multiple attributes for a given location (spatial-multivariate). Covering the entire field would be a bit too ambitious for this thesis. We therefore restrict ourselves to spatial patterns. In this chapter we briefly discuss the background and previous work within the area (following the global structure as depicted in Fig. 2.1).

The need for scalability. Exploration of large volumes of information, in limited space, requires visualizations at different levels of detail [38]. Common approaches are to display a static set of visualizations at different levels of detail or by making the visualization interactive, by including the user in the process. The latter means that users can also benefit from their creativity and knowledge of the data [25]. We split this chapter into two parts, the first discusses related work on different visualization techniques (Section 2.1) and the second on the scalability of the visualization (Section 2.2).
2.1 Visual representation

There are a lot of techniques available to visualize point sets distributed over the plane. In this section we will discuss some of the more common ones.

2.1.1 Variations on scatter plots

Aside from their shortcomings (cluttering and overplotting if the number of points is large), scatter plots are still widely used and have proven to be effective for analyzing at least moderately sized datasets [48]. Because of the need to be able to cope with a larger amount of points, a number of techniques have been proposed that address these shortcomings [6] without resorting to more drastic measures such as contour maps. A general taxonomy on clutter reduction in information visualization is given by Ellis and Dix [17], who outline a number of available methods. In this section we describe some of the more common solutions related to scatter plots.

![Illustrative examples of techniques to reduce clutter in scatter plots](image)

Figure 2.2: Illustrative examples of techniques to reduce clutter in scatter plots: (a) original, (b) displacement, (c) sampling, (d) reducing the opacity, and (e) using a smaller point size.

Note that these solutions do not scale very well as the number of points grows.

**Displacement.** This method tries to reduce visual occlusion caused by overplotting by displacing points by a certain distance around their original position (Fig. 2.2(b)). Trutschl et al. reason that shading, transparency or remapping effectively portray the crowding of points, but fail to give insight into the relationships between neighboring points [44]. They propose a neural-network based algorithm that organizes overlapping points by displacement.

**Filtering and sampling.** Selectively picking points (by filtering), or simply sampling the entire collection of points, can greatly reduce cluttering and overplotting (Fig. 2.2(c)). A common type of map where sampling is applied is the dot map [40], where a single dot represents a number of occurrences of the phenomena that is shown. Algorithms exist to do this automatically, for example, by de Berg et al. [14].

**Visual Attributes.** A simple, yet effective technique, is to change certain visual attributes of the points, e.g., opacity (Fig. 2.2(d)) and point size (Fig. 2.2(e)). Reducing point size directly reduces clutter but also possibly makes the scatter plot less readable. Reducing the opacity creates a sort of density effect, indicating cluttered regions by means of a darker shade. However, reducing the opacity too much also makes the scatter plot less readable, because single points will be less noticeable. If we choose an opacity that is too high, dense regions will still contain overplotting (not visible due to clamping). Picking one that is too low will result in individual points that are barely visible. Reducing the point size has similar problems: a smaller point size means less overplotting but makes individual points less visible, and also, coinciding points still lead to overplotting.

**Distortion.** Deliberately distorting the projection area of a scatter plot can reduce cluttering in regions that contain a lot of detail (i.e., a lot of points). Expanding these crowded regions (and as a result shrinking the sparser regions) will automatically reduce cluttering because the distortion will - to some extent - pull apart overlapping points. Typical examples of this are cartograms [16].
2.1.2 Grouping, aggregation and clustering

There are a number of methods that in some way group, aggregate, or cluster the input set of points and then visualize this on a map. In geographic visualization we can find a lot of examples that are as a whole often referred to as thematic maps (Definition 1).

**Definition 1.** Thematic maps consist of a base map (e.g., administrative boundaries) and a number of overlay layers that depict specific themes related to a geographic location.

In our case, we only consider univariate thematic maps (maps that only depict a single attribute for the given region), because we only assume input that is spatial univariate data. In this section we list a few of the more common types of (univariate) thematic maps.

![Thematic maps examples](image)

**Figure 2.3:** Illustrative examples of different types of thematic maps showing population density: (a) choropleth, (b) proportional symbols, and (c) clustered regions.

**Choropleth mapping.** In choropleth mapping a subdivision of administrative regions (sometimes called enumeration units) is assumed to be available. These regions, e.g., countries, states, city blocks, are colored or textured to visualize a certain statistical attribute for that region (Fig. 2.3(a)). Often regions on the map are classified, i.e., each region is assigned to a class and color is used to indicate membership. Well known examples are election maps showing the majority vote for each region on the map. Unclassified versions of choropleth maps are possible as well, these visualize a more continuous statistic like total population count or average income. The shade of a color is used to indicate value and a legend is attached to indicate the scale.

Choropleth maps can also be extended to multivariate display (visualize multiple attributes) by using a combination of color and texturing of regions.

**Aggregate data and proportional symbol maps.** In proportional symbol maps (Fig. 2.3(b)), data associated with a certain location (for example cities and their population) is aggregated and visualized using proportional symbols [34]. Besides aggregation proportional symbol maps can also be used to visualize more discrete phenomena, a commonly known example being maps showing the epicenters and magnitudes of earth quakes.
Dasymetric mapping. Choropleth maps are best suited for display of information specific to the administrative regions themselves. However, it is not always the case that the underlying phenomena of the data will follow the exact boundaries of these regions. As a solution dasymetric maps are proposed. Regions are determined using ancillary data and do not necessarily correspond with any administrative boundaries. For example if we want to visualize the population of a certain animal species on a map, it makes sense to choose regions that represent their area of natural habitat, instead of simply using the administrative regions.

Clustering. Besides using administrative regions (choropleth/proportional) or ancillary data (dasymetric), another approach is to try to determine the regions from the set of points itself (Fig. 2.3(c)). A straightforward approach to do this is by clustering the set of points. Clustering methods are usually divided into two groups: partitioning and hierarchical approaches. Partitioning approaches aim to partition the entire input in a set of non-overlapping clusters, while hierarchical approaches decompose the entire input in a set of nested clusters. The latter usually store the output in a tree structure called a dendrogram, where each node represents a cluster and traversing up the tree represents clusters merging. Because spatial data is often naturally hierarchically structured (countries are formed out of provinces, which in turn contain cities, etc.), hierarchical approaches are often the first choice for clustering spatial data.

2.1.3 Contouring

Contour maps display lines of equal value (called contours or isolines) for a scalar field (in our case representing the point density). Typical examples are contours plotted on top of a continuous density map (Fig. 2.4(a)), or more discrete versions, where the interior of a contour is uniformly colored (Fig. 2.4(b)).

In geographic visualization maps that depict smooth continuous phenomena are called isarithmic maps. Despite the fact that the phenomena are assumed to be continuous, the input for isarithmic maps itself is discrete. This is because it is impossible to measure and store the phenomena truly continuous. Often data is sampled on a grid, or at certain specific points only (called control points).

Several techniques exist to reconstruct the continuous signal from the values only available at the sampled points. In geographic visualization methods to interpolate between neighboring observation points are often used. Examples are Inverse-Distance interpolation and Kriging [15, 40]. In statistics and data analysis, a more common approach is density estimation, which is the construction of an estimate of the unknown probability density function of a sample set of observation points [39]. Using a kernel estimator, often referred to as Kernel Density Estimation (KDE), is a popular way to compute density estimations used for density plots.
2.2 Scalability

Visualization of data at different levels of detail introduces conceptual issues such as what features do we show for a given scale and in how much detail do we show them. These issues are well recognized in digital cartography, known as cartographic generalization.

Scalability is important if we, for example, want to implement interaction methods such as panning and zooming. When doing this, it is important that changes are gradual and predictable such that the user does not lose context. Because visualizations are automatically generated, this poses challenges such as what to show as well as on-the-fly computation. These issues have been recognized in areas such as map generalization in digital cartography [37].

**Cartographic generalization.** The purpose of cartographic map generalization is to produce a map at reduced detail (from the most detailed data) appropriate to purpose and scale of the map. On large scale maps high detail can be shown, but on smaller scales and within the same available space, reducing this detail (e.g., by removing or simplifying features) is necessary to keep the map legible. A general definition is given by McMaster et al. [29]:

**Definition 2.** Digital generalization can be defined as the process of deriving, from a data source, a symbolically or digitally-encoded cartographic data set through the application of spatial and attribute transformations. Objectives of this derivation are: to reduce in scope the amount, type, and cartographic portrayal of the mapped or encoded data consistent with the chosen map purpose and intended audience; and to maintain clarity of presentation at the target scale.

Cartographic generalization has proven to be a difficult and very subjective problem that is hard to automate. To better understand the complexity and to be able to automate (parts of) the process, the problem is commonly divided into smaller tasks called generalization operators. Each of these operators describe tasks on certain features of the map originally done by cartographers by hand. Examples are simplification of line segments (e.g., rivers, roads, or boundaries) and symbolization of important features that become too small to be represented on the map. Some of these generalization operators are relevant to our research question. In the next paragraph we briefly discuss some of the relevant operators.

For example, the typification operator aims to turn an initial set of objects into a subset of representative objects that still preserves the structure of the original set. Algorithms by Burghardt and Ceconi [10], and Regnauld [35] have been developed that can generalize individual houses into blocks of buildings. As a preprocessing step for the typification operator graph based clustering methods have been proposed to help find higher level structure in sets of objects, for example, by Anders and Sester [3]. Relevant to the shape of a region are the area simplification and schematization operators, for example, see the work of Barkowsky et al. on simplification of geographic shape [5]. Slocum et al. list a few other relevant fundamental operators, e.g., aggregation, where individual point sets are grouped and represented as areal features, and amalgamation, which is the process of merging multiple areal features into a single areal feature [40].

**Alternative interaction methods.** Besides panning and zooming there are a number of other interaction methods [24]. One method that is often used in information visualization is the brushing and linking technique. This technique links different visual representation of the same data together. This can be the same data using different visualization techniques (e.g., scatter plots linked to histograms), or the same data at different levels of detail. Brushing and linking means that the different visualizations are linked together, such that selecting certain features or changing properties in one visualization propagates to the other visualizations.

Other techniques like distortion allow the user to quickly inspect certain regions of the visualization by showing more detail, for example using a fish-eye view.
Chapter 3

Problem Analysis

Recall the problem statement as given in Chapter 1. From the input set of points we want to automatically discover regions and display these on a map. In this chapter we explore the different aspects of this problem (as summarized in Fig. 3.1) in more detail, and define the scope of this project. We first make three observations on standard topographic maps and the city metaphor that serve as the foundation for discussion in the remaining part of this chapter.

Observation 1. Regions (such as cities, green-land, and industrial areas) are displayed as simple shapes with uniform colors and crisp boundaries, sometimes labeled and accompanied by a symbol hinting its size.

Observation 2. Regions are more or less uniformly distributed over the map according to their local importance, e.g., it is possible that a relatively small city in a sparse area is shown while a larger city in a denser area is not.

Note that this distribution does not only depend on the regions itself, but on the scale, physical size and function of the map as well. Looking at a country as a whole, we might only expect to see the most important cities, but when we zoom in, eventually, we would like to see each individual person. The same principle should hold in information visualization in general, as compactly put by Shneiderman [38]: Overview first, zoom and filter, then details-on-demand. As a result of this, we should be able to generate a visual representation at different levels of detail, to which we refer as the scalability of the model (e.g., for animation and interaction purposes). Smooth and predictable transitions between different levels are important such that the user does not lose context. Based on this we make a third observation.

Observation 3. Depending on what the user wants to see (e.g., an overview, or a detailed version of a small region of the map) we should present different levels of detail. Ideally transitions between these levels are smooth and predictable.

We aim at automatically producing maps that adhere to these observations, ideally with as little parameters as possible. We first discuss why alternative choices have not been pursued (Sec-
3.1 Alternative Choices

In the previous chapter (Chapter 2) we have outlined a number of methods that we can use to visualize large numbers of points distributed over the plane. The three observations made in the introduction to this chapter are in favor of an approach that involves clustering of the points. To get a clearer picture of the advantages and disadvantages of the type of map that we pursue, we briefly revisit the options from the previous chapter, and compare them to maps that adhere to our observations.

**Scatter plots.** In Chapter 2 we have discussed ways to reduce visual cluttering and overplotting in scatter plots, using techniques such as changing visual attributes, displacing points or filtering points. Trutschl et al. note that the more aggregated approaches, scatter plots using transparency or remapping (and we also consider density maps here), fail to give insight into the relationships between neighboring points [44].

![Figure 3.2: Illustration of the effect of overplotting in scatter plots. All four depict the same data: locations where pictures were taken in the city center of Paris: (a) scatter plot, (b) scatter plot (where the points are drawn 10% opaque), (c) density plot on linear scale, and (d) density plot on logarithmic scale.](image)
Figure 3.2 illustrates this by showing four versions of the same datasets, locations where photos were taken in the city center of Paris. Because of our knowledge of the data we can distinguish two different types of features; tourist hot spots (e.g., the Eiffel tower, the Louvre museum and the Notre-Dame Cathedral), and patterns revealing the infrastructure such as streets and bridges. Directly plotting the points (Fig. 3.2(a)) clearly shows the street patterns, but results in overplotting at the tourist hot spots. Trying one of the methods described in Section 2.1.1, e.g., drawing the points at 10% opacity (Fig. 3.2(b)), reduces this, but it is still impossible to judge the density, as there is no guarantee that all overplotting has been removed.

One way to get this guarantee is by plotting the density distribution of the points instead. Using a linear scale (Fig. 3.2(c)), we see that the hot spots dominate the map. Because of the high variation in density (between hot spots and other prominent patterns, like the street network), the density map with linear scaling shows little resemblance with the original set of points. In such a scenario it is common to apply a logarithmic scale instead (Fig. 3.2(d)). An interesting question is which depiction is right.

A recent comparison by Tory et al. shows that contour maps and landscapes are not more memorable compared to simple dot maps [43], however, they use a relatively small number of points (up to 1000). We assume larger numbers (up to one million points). Within limited space, using displacement techniques on points in crowded regions would quickly become infeasible. Moreover we consider the underlying spatial patterns to be most important. Relationships between individual points (especially on large data sets) is of less importance.

To conclude, for large amounts of points, the scatter plot has problems that are not easily overcome (e.g., using techniques as described in Section 2.1.1). Clustering points instead, allows us to define an ordering on the clusters of points, which means we can explicitly choose the amount of detail we want to show, and therefore, explicitly remove cluttering and overplotting.

**Crisp regions vs. fuzzy density maps.** We state that we aim to use crisp region shapes (Fig. 3.3(a)) inspired by standard cartographic maps and the city metaphor. Why is this any better than simply plotting the density map (Fig. 3.3(b))? This question cannot be answered easily because going either way might have advantages depending on the type of data and the goal of the data exploration.

![Figure 3.3: Population density of the Netherlands: (a) clustered regions, and (b) grayscale density plot (on linear scale).](image-url)
Discretization from continuous density maps into crisp regions means losing information. When we ask the user to estimate the density at a specific location on the map, the density map itself is clearly superior, because at any location the density is coded using either color, elevation or some other property. However, the problem with these fuzzy density plots is that they are harder to interpret than a more schematic map, for example, one that shows only hot spots as simple crisp shapes. Asking users to pinpoint hot spot areas, or important regions, will probably give different results for different users. Users in general will have different opinions on what exactly is defined as a hot spot, and what shape it should have.

By explicitly only showing these important regions, we can try to relieve the user from this task, although doing this automatically is challenging, because there are no rules that tell us what constitutes a region. Discretization into regions means that we have direct control over the regions that we show, and hence, also over the amount of detail shown on a map. We have explicitly chosen for the discrete approach, but we realize that for some scenarios we are better off using a plain density map. Therefore, for our model we choose to optionally show density maps as well, possibly in combination with crisp regions.

Using a third dimension. Instead of using two dimensions and color coding, we can also use the third dimension (elevation) to depict scalar value. This is sometimes referred to as the landscape metaphor. In literature it is often said that people are naturally trained in understanding the topological structure of such landscapes [50]. However, in information visualization, 2D is often preferred over 3D. This is because the latter suffers from problems such as occlusion, scene complexity (often more difficult to comprehend than 2D), depth ambiguity and requires additional complex interaction such as 3D scene navigation [19, 31, 39, 43].

Although, we use some techniques generally considered more than 2D (sometimes loosely referred to as 2.5D), e.g., shading (Section 5.5.3) and color coded density maps, our main focus is on the 2D depiction of the regions.

3.2 Input

We expect the input \( P \) to be spatial univariate point data, formally \( P \subset \{(x, y, w) \mid (x, y) \in \mathbb{R}^2 \land w \in \mathbb{R} \land w \geq 0\} \), where \( x \) and \( y \) are the point’s coordinates in the plane and \( w \) denotes its weight. Spatial data is data that has some relation to geographic location, also referred to as georeferenced data. Univariate means that the data only has a single attribute, e.g., income, age, temperature (the weight \( w \) specifies this attribute).

Specific research exists on visualizations of multivariate data (multiple attributes) and temporal data. These focus on visualizing relations between different attributes, or how patterns change over time. Often there is no clear distinction between the different exploration types, e.g., temporal analysis is possible using animation over time, and multivariate analysis by using methods that reduce the dimensionality of the input (spatializations). We restrict ourselves to univariate data at a single point in time.

Spatializations. By using techniques such as MultiDimensional Scaling (MDS) [7] it is possible to project multivariate data to a lower dimensional space, such as a plane. This way it is possible to also visualize these in a spatial way. Such visualizations are called spatializations [19] and are also applicable to our model.

Map projection. For most real world data we expect the points to be given using the World Geodetic System (WGS) standard: spherical coordinates longitude and latitude, which specify a unique location on the Earth’s surface. As a result of using such a spherical coordinate system, before we can display the points on screen, we need to project them onto the plane. This process is called map projection. Many different types of projections exist, for an overview we refer to Snyder [41]. We use the spherical Mercator projection, which is explained in more detail in Section 5.1.
Spatially sampled data. Not all phenomena can be captured in full detail using only a discrete set of points, for example, phenomena that are naturally continuous (e.g., temperature). Also, because of more practical limitations, such as the resulting size of the dataset, or the dynamic nature of the data that we want to capture (e.g., population). A common strategy is to reduce the resolution of the data by sampling the phenomena at specific control points (e.g., in a grid). Slocum et al. distinguish two types of sampled point data [40]: true point data, for phenomena that can be truly sampled at any point \( p \) in the plane (because they are continuously defined over the plane); and conceptual point data, for phenomena that cannot, and have to be sampled by aggregating over larger areas (e.g., taking the total or average within a grid cell).

3.3 Region discovery

In Observation 2 we state that regions should be more or less uniformly distributed over the map according to their local importance. This statement alone is somewhat vague. Automatically finding such regions poses a challenge. Intuitively, we prefer larger (or more important) regions over smaller regions. This indicates that there should be some sort of ordering in the regions, e.g., we can use the weighted point count, total area, or a combination of both to define the region’s importance. However, such a straightforward approach does not always produce correct results. For example, we might decide to hide a city that is surrounded by a number of bigger cities (i.e., have a higher population count), while in another region of the map, it is possible that we show cities that are actually smaller than the hidden city. In other words, there is a notion of local (or relative) importance.

This problem is closely related to the cartographic problem of settlement selection [46].

Outliers. As opposed to identifying common patterns in the data, in some cases it might be more interesting to locate the rare instances that do not fit these patterns: outliers. When clustering or otherwise grouping points in some fashion, the outliers are the points not contained in any of these clusters or groups. While for some types of data these outliers can be considered noise, for most real data they have actual meaning, and we have to consider what to do with them. We can simply choose to omit these outliers, or show them all individually (e.g., like a bagplot [36]).

Besides considering being an outlier as a binary property, it is also possible to assign a degree of being an outlier to each point. An example of this is the Local Outlier Factor (LOF) [8]. This measure indicates the degree of isolation of each point within the set of points. Instead of simply showing all outliers, which for large point sets will still result in a lot of cluttering, this measure can be used to only show the interesting outliers.

Although the latter is an interesting problem, we focus on the clusters itself, and only provide the option to either show or hide all outliers. The outliers are hidden by default (as on a standard topographic map), because we assume large sets of points as input, which will result in a lot of cluttering if we show all outliers.

3.3.1 Scalability

So far, we have discussed aspects of the problem with a strong focus on the visual representation of a single level of detail. This makes sense for a static map, however, if we want to generate visual representations at multiple levels of detail, e.g., by user interaction, or animation, we should look at different scales well. As per Observation 3, ideally, we want these transitions to be smooth and predictable.

Animation. For exploration at multiple levels of detail, or of patterns in data over time, animation (as we zoom in on a region, or vary time) can give insight in how regions evolve (as in grow, split or merge with other regions).
**Interaction.** In the process of data exploration interaction is an important tool. In Section 2.2 we have outlined several of these interaction techniques. Most GIS and mapping software offer interaction in the form of view manipulation, e.g., browsing through different map regions (panning) or changing the scale (zooming). This way, we give the user a lot of freedom, because it is possible to practically view any region of the map at any scale.

Our aim is to be able to automatically generate these animations (off-line). On-the-fly interaction by panning and zooming (with smooth and predictable transitions) would be the ideal goal, but generation of a single visual representation within our model is not nearly fast enough to do this real-time. For future work (Chapter 7), we sketch an idea on how to extend our model to allow this type of interaction by precomputing different levels of detail and morphing between intermediate temporal visualizations.

### 3.4 Visual representation

#### 3.4.1 Region properties

In this section we discuss some of the individual properties of regions. Addressing the question *What do we expect regions to look like?* gives subjective results and depends on user preference as well as the function of the map. Varying these aspects (e.g., shape complexity) gives a range of possibilities. We enumerate several of these aspects, give different variations, and by doing this try to make the range in variation for the type of map that we pursue more explicit.

**Shape complexity and compactness.** The *complexity* of a region shape is a trade-off between accuracy and simplicity of the map (Fig. 3.4). Using very complex shapes we can depict most of the geometric properties of the set of points, but this also means that we obtain a map that contains a lot of detail and might be harder to interpret. Using simpler shapes, we lose some of the detail in the original data, but we get a map that is hopefully aesthetically more pleasing and easier to interpret. Several measures for shape complexity exist [26].

We can also generalize the complexity to multiple shapes (*compactness*): a number of smaller, densely packed shapes, create a complex composition, while we can also represent them as a whole using a single simpler shape (Fig. 3.5).

One way to control the complexity is to explicitly measure it for each individual region and adapt regions accordingly. A simpler approach is to use a density map, derived from the input set of points, and to control the complexity with operations on this density map (e.g., smoothing).

**Region vagueness.** When going from fuzzy to very crisp regions (Fig. 3.6), we have to make a decision where we place the boundary of the region. Using crisp boundaries might indicate to the user that they are precisely defined and that we are certain about their position, while this is usually not the case. We might consider all points above a certain density to form a region, which immediately raises the question, what density value we should use? And even if we find a good value, we might get an equally valid region by taking just a little lower value which encloses more points.

Besides determining the right placement, we might also try to solve this problem by not using crisp regions, but somehow indicating how certain we are the boundary is to be placed at that position. The density map gives us some clues; areas with a very rapid change in density (a steep slope) indicate a high probability that a region ends there. On the contrary, large
areas, with less variation in density (a gradual slope), indicate that the boundary is equally likely to be anywhere in that area. Some straightforward methods to visualize this are by using fuzzy boundaries (the most extreme example being the density map itself), or by varying properties such as thickness, color, or crispness of the boundary. For example, we could use a very thick boundary if we are certain about the position of the boundary, and a very thin one if we are not.

**Nested regions.** Originating from the idea that we want to visualize the uncertainty, besides using fuzzy or crisp boundaries, we can also use a number of intermediate boundaries (or from our point of view: nested regions) that indicate a degree of membership (Fig. 3.7). Examples of these are the egg-yolk model [13] and the bagplot [36]. The density value of a point can be seen as a the degree of membership. The egg-yolk model considers a region to exist of two contours. The first, the inner contour, indicates all points that are certainly inside the region, and the second, all points that are possibly in the region. All points not in the contours are considered to be definitely not inside the region.

**Minimal size and distance.** The notion of minimal size and minimal distance is related to the complexity of a shape. Looking at individual regions we might impose a minimal size (area) threshold, because up to a certain size, regions might simply be too small to be recognized. For a single region there is also a notion of minimal distance, e.g., very sharp features on the boundary of a region might better be replaced by a smoother boundary.

**Region overlap.** In theory, when we use regions (covering groups of points), there is nothing that forbids us to create regions that overlap each other. For example, this can happen when we explicitly choose regions that contain the same subset of points, or if we somehow transform the region shape such that this happens. However, since one of our concerns with scatter plots is the overlapping (or overplotting) of individual points, it makes sense that we look for a solution that prohibits overlapping of individual regions as well.

### 3.4.2 Other map features

A standard topographic map usually exists of several layers containing different map features, e.g., rivers, roads, borders and municipalities. These features are important as their characteristics help the user to identify locations and recognize parts of the map. Automatic selection and reduction of the detail of these features, appropriate to the scale of the map, is a classic problem in digital cartography called cartographic generalization [37].

**Spatial context.** We call the administrative boundaries, rivers, and roads, the spatial context. In thematic maps these features are often called the base map. Showing this base map improves the aesthetics as well as the effectiveness (in recognizing features) of the map. However, it might also give undesired results. Small measurement errors, or simply properties of the algorithms (e.g., over-smoothing in KDE), can cause regions to overlap with the base map (Fig. 3.8). For some types of data (e.g., population) this can give undesirable results, for example, if parts of a region are positioned at impossible locations (e.g., over water).

**Symbolization.** On standard geographic maps, the shape of a region usually represents geographic properties (e.g., the administrative boundary or total area), and not necessarily its size as in the number of contained points. To indicate this, symbols are used. Cities are classified according to their population count (e.g., large, medium sized, and small cities) and a symbol indicating class membership is plotted on top (or in the vicinity) of the region shape on the map. These symbols are usually abstract, for example, a filled square or circle, and proportional to the class size (i.e., a bigger symbol indicates a larger population count). Challenges are to position these symbols and determine their size and variation in size.
In this chapter we develop a model based on the discussion in previous chapter. We combine a collection of new and existing methods and data structures that we can use to generate maps, and experiment with the different aspects of the problem as described in Chapter 3. Implementation details are given in Chapter 5.

Figure 4.1 summarizes the important components of the model. From the input set of points $P$ we generate a triangulation $\mathcal{DT}$ (Section 4.1) and a density map $\mathcal{F}$ (Section 4.3). Based on these, we generate the contour tree $\mathcal{T}$, which is a graph structure that describes the topography of the density map $\mathcal{F}$ (Section 4.4). From the contour tree $\mathcal{T}$, we select a number of nodes - that represent regions - to be displayed (Section 4.5). Using the density map we generate a number of map layers, most importantly, the layer containing the visual representation of the regions. The final map is created by merging these layers.

![Diagram of model components](image.png)

*Figure 4.1: Important components of the model.*

The individual components, as illustrated in Figure 4.1, are discussed in the remaining part of this chapter. The rendering step is added for completeness sake and is discussed in Section 5.3.
4.1 Triangulation

The algorithms we use require ancillary data structures such as proximity graphs, triangulations and (simplicial) meshes (Definition 3) defined over the input data $P$. Any planar triangulation is a simplicial complex, and can be considered a proximity graph as well. Therefore, we can compute the triangulation of $P$ once, and use it for all the algorithms that require one of these data structures.

We have chosen to use a Delaunay triangulation $T$. These have some favorable properties, such as, a maximized smallest interior angle of any triangle in $T$ (i.e., avoid skinny triangles). We compute the triangulation using Fortune’s $O(n \log n)$ sweep-line algorithm (see the original work for details [21]).

**Definition 3.** An $n$-simplex is the simplest possible polytope in $n$-dimensional space. For example, in $2D$ this is a triangle. An $n$-dimensional mesh $M$ is said to be a simplicial complex if it solely consists of $n$-simplices.

4.2 Clustering

When we are trying to find regions within the input set of points $P$ (i.e., clusters), the first obvious approach would be to use existing clustering methods. We consider scalability to be an important factor. This means that the clustering should be dependent on the scale. Using a standard (partitioning) approach means that we not only would have to recompute the clustering for each scale, but also, there is not necessarily a clear relation between clusters at different scales.

Therefore, we use an hierarchical (or agglomerative) clustering approach. Instead of generating a single set of clusters, these types of algorithms produce a tree structure that represents a hierarchy of clusters, called a dendrogram. A branch starting at node $n$ in this tree represents a cluster that contains all children in the subtree starting at $n$. A node with multiple children indicates that a cluster is split into multiple clusters. The advantage of this type of methods is that we can precompute the dendrogram and based on the requested scale we can find the appropriate depth to select clusters.

We have implemented the following two clustering schemes: Hierarchical Parameter-free Graph Clustering (HPGCL) by Anders [2] and AMOEBA by Estivill-Castro and Lee [18]. Both algorithms follow the same global strategy:

- Generate a proximity graph (e.g., a Nearest Neighbor Graph (NNG) or a Delaunay Triangulation (DT));
- Add each point in the input as a leaf (each point defines a cluster);
- Construct the dendrogram from bottom-up, by merging neighboring clusters (from the proximity graph) if they meet certain requirements.

Both methods apply different metrics for merging neighboring clusters. Details can be found in the respective papers.

However, experimenting with the results of the two methods did not give satisfactory results. Traversing through the dendrograms, as generated by both schemes, results in abrupt changes in cluster size as well as shape. On top of this, the effect of changing parameters on the resulting number of levels in the dendrograms is hard to predict. This is not desired, as we want to have gradual and predictable transitions between different levels in the three. Although, we can possibly get better results for individual datasets by fine-tuning the parameters, we choose to take a different approach that is based on density maps and contour trees.
4.3 Density maps

In previous sections we have reasoned that clustering the points gives undesirable results. A second idea is to use the density distribution (we call this the density map) of the set of points instead (Fig. 4.2). The contours (or isolines) at a given density \( d \) (visually) create clusters of points that have a density higher than or equal to \( d \). These contours can be seen as the intersection of the density map with a plane at density \( d \) (as illustrated in Fig. 4.3). Intuitively, if we vary this density \( d \) (moving the plane that intersects up and down), topographic changes such as the creation, collapsing, merging and splitting of contours will be very gradual, smooth and predictable, at least for small changes of \( d \).

The problem is that by drawing the contours we only visually create clusters. Ideally, we want to have explicit clusters, such that for a given cluster we know exactly which points it contains. For this we use a structure that compactly describes the topography of the density map, called the contour tree (Section 4.4).

Another advantage of using the density map over hierarchical clustering approaches is that we can use the density map for the actual rendering of regions as well (Section 5.3).

**Definition 4.** A scalar field \( F \) is a pair \((f, D)\), where \( f \) is a function that assigns a scalar value to each point in the domain \( D \).

We define a density map, formally, to be the scalar field \( F = (f, \mathbb{R}^2) \) (Definition 4), where \( f(p) \) denotes the density for a given point \( p \) in \( \mathbb{R}^2 \). This definition requires the densities to be defined at any point in \( \mathbb{R}^2 \), however, for the input \( P \) this is only defined at the vertices of \( P \) itself. As already mentioned in Section 2.1.3, we can estimate these unknowns by using Kernel Density Estimation (KDE).

**4.3.1 Kernel Density Estimation**

Kernel Density Estimation is a statistical technique [39] that can be used to estimate the density distribution of an observed sample set of data. It works by convolving each observation point with a kernel (Fig. 4.4). In essence, each observation point has a certain contribution to the overall density. This contribution is determined by a kernel function and a kernel bandwidth, commonly referred to as \( K \) and \( h \) respectively. In general, the higher the bandwidth, the more smoothing of data will take place. Depending on the context, we will sometimes also refer to the kernel bandwidth as the radius of the kernel.

In Section 2.1.3 we have briefly discussed some of the alternatives to KDE, e.g., Inverse-Distance interpolation and Kriging. These can be used to estimate the density at unknown points as well. However, we choose to use KDE instead. The main reason is that varying the amount of smoothing of the KDE estimate is very straightforward by varying the kernel bandwidth.
4.3. DENSITY MAPS

The estimated density \( f(p) \), for a so-called evaluation point \( p \in \mathbb{R}^2 \), is computed by taking the mean of the contributions of all observation points in the input \( P \). More formally,

\[
f(p) = \frac{1}{|P|} \cdot h \sum_{q \in P} K(u),
\]

where \( K \) is the kernel function, \( h \) the bandwidth, and \( u \) denotes the normalized distance of observation point \( q \) to the evaluation point \( p \):

\[
u = \frac{\text{distance}(p,q)}{h},
\]

given a distance function \( \text{distance} \).

Note that in our case \( P \) is not a regular set of points, but each observation \( p \in P \) optionally has a weight \( w(p) \) as well. We incorporate this into the KDE estimate by multiplying the contribution of an observation point \( p \) by its weight. This is equivalent to the KDE estimate we get when we simply consider the weight of a point \( p \) to be the number of observations that occur exactly at the position of \( p \). The new estimate \( f(p) \) is given by

\[
f(p) = \frac{1}{W} \sum_{q \in P} K(u) \cdot w(q),
\]

where \( W \) is the total weighted point count of input \( P \).

In literature, we often see a Gaussian kernel function \( K_G \) in dimension \( d \) of the form

\[
K_G(u) = \frac{1}{h^{2d/2}} e^{-\frac{1}{2} u^2}.
\]

However, a problem of the Gaussian kernel is that it has infinite support. For computational purposes we require our kernel function to be zero (no contribution) for every point outside of the kernel radius \( h \). To achieve this we use a different kernel, the Triweight kernel. This is an approximation to the Gaussian kernel and satisfies the requirement of having no contribution outside of the radius \( h \). The Triweight kernel function \( K_T \) is defined by

\[
K_T(u) = \begin{cases} 
\frac{35}{32} (1 - u^2)^3 & \text{if } u < 1 \\
0 & \text{otherwise.}
\end{cases}
\]

In terms of a kernel function \( K(u) \), we can express the total contribution \( C \) of an observation point to the density map as

\[
C = \int \int K(u) \, dx \, dy,
\]

where \((x, y)\) denotes a point in kernel space.

Just as for the Gaussian kernel, for the Triweight kernel, this contribution \( C \) is independent of the bandwidth \( h \). This means that even if we use a variable bandwidth (Section 4.3.2), or a distorted kernel (Section 5.2.1), it is guaranteed that each observation point still has an equal contribution to the overall density.

The technical details on the computation of density maps are given in Section 5.2.1.
Optimal bandwidth selection. For KDE it is common practice to automatically determine an optimal bandwidth [23]. Optimal is usually defined as minimizing a certain error function (i.e., the estimates at observation points are close to the real value). This is useful if we require a (provable) best estimator that fits the original signal as well as possible. However, in our case this is less important. We want to experiment with the bandwidth. In some cases we even deliberately over-smooth the data to reduce the complexity of the region shape, or to make sure important regions will be more prominent.

4.3.2 Variable bandwidth kernels

In standard KDE a fixed bandwidth is used for each observation point. This can result in under-smoothing of sparse areas, and over-smoothing of dense areas of the data. A solution is to vary the kernel bandwidth for each individual point, usually such that a smaller kernel is used for points in denser regions. These type of KDE methods are said to be adaptive, or variable. We have implemented two adaptive kernels; a kernel based on a proximity graph over the input $P$ and a kernel for geographical point data by Brunsdon [9].

Using the triangulation. Any proximity graph defined on the vertices of input $P$ allows us to quickly find neighboring vertices of a vertex $p \in P$. Using this information we can make a crude estimate of the density at $p$ and use this to change properties of the kernel. For this estimate we use the mean weighted edge length of all the neighbors of $p$ (denoted by $mwe(p)$):

$$mwe(p) = \frac{1}{|N|} \sum_{n \in N} w(n) \cdot \text{distance}(n, p), \quad (4.7)$$

where $N$ is the set of neighboring vertices of $p$, $w(n)$ is the weight of a neighbor $n$ and distance is a distance function. This follows our intuition; in denser regions of the graph points will be more closely surrounded by their neighbors than in sparser regions.

Using the Triweight kernel, as defined in the previous section, we scale the bandwidth for an observation point $p$ with the mean weighted edge length estimate $mwe(p)$. We can rewrite the KDE density estimate (Eq. (4.1)) to

$$f(x) = \frac{1}{Wh} \sum_{q \in P} K\left(\frac{1}{mwe(p)} u\right), \quad (4.8)$$

where $u$ is defined as in Equation (4.2). Informally this translates to more smoothing in sparse regions of the dataset and less smoothing in denser regions.

We use the Delaunay triangulation $T$ as a proximity graph for this variable kernel. This type of triangulation has some useful global properties, such as, a maximized smallest interior angle of any triangle in $T$ (avoid skinny triangles, i.e., neighboring vertices in the triangulation are really in the vicinity). An additional advantage is that we have already computed the Delaunay triangulation of the input $P$ (see Section 4.1), and as such, can reuse it here.

Note however that this type of variable kernel is less useful if the input is sampled on a grid. This means that besides the weight of the points, the actual triangulation is simply a triangulated grid, and does not say anything about the underlying distribution of the density.

Brunsdowns’ adaptive kernel. Based on the same principle, but using a slightly different approach, is Brunsdowns’ adaptive kernel. It requires two passes. In the first pass a density map is computed using a fixed kernel bandwidth. The second pass is made adaptive by scaling the radius with the estimate from the first pass.
This scaling factor \( l_p \) is defined as

\[
l_p = \left( \frac{f^*(p)}{h} \right)^\alpha,
\]

(4.9)

where \( f^*(p) \) denotes the fixed-bandwidth density estimate computed in the first step. Brunsdon uses a method based on likelihood cross-validation to find optimal values for the fixed kernel bandwidth as well as the \( \alpha \) parameter (see the original work for details [9]).

For an observation point \( p \) a new kernel bandwidth \( h_p \) is defined in terms of the scaling factor \( l_p \) \((h_p = l_p h)\). The KDE equation (Eq. (4.1)) now takes a slightly different form:

\[
f(p) = \frac{1}{W} \sum_{p \in P} \frac{1}{h_p} K(u),
\]

(4.10)

where \( u \) is redefined in terms of \( h_p \) as

\[
u = \frac{\text{distance}(p, q)}{h_p}.
\]

(4.11)

### 4.4 Contour tree

The contour tree was used by Bajaj et al. to compute minimal seed sets to efficiently trace contours [4], and by Oesterling et al. to visualize structure in arbitrary dimensional point clouds [32]. In general, a contour tree can be used to find the contours of a scalar field, or simply to find the scalar values at which topological changes occur. The contour tree proves to be a powerful tool, which can be applied in many different scenarios.

We use the contour tree \( \mathcal{T} \) primarily as a way to access (or traverse) the topology of the density map. The density map itself contains an infinite number of possible contours, and would therefore be infeasible to use directly. The contour tree simplifies this by allowing us to only traverse the contracted contours of the density map. We consider all the contours described by \( \mathcal{T} \) to be possible regions that can be displayed.

![Contour Tree Diagram](attachment:image.png)

Figure 4.5: Example of a contour tree \( \mathcal{T} \) for density map \( \mathcal{DM} \) (visualized using a contour map and a height field (b)).

The contour tree is best explained when we view the density map as a 3D landscape [50], where the elevation indicates density. Specific densities can be highlighted using contours. These contours...
can be seen as the intersection of the landscape with the plane at a given density value (Fig. 4.5(b)).

As we move this plane up and down, we will see contours appearing, disappearing, merging, or
splitting. The contour tree is a graph structure that compactly describes these topological changes.

Before we give a more formal definition of a contour tree, we first briefly discuss some of the
computational details. This will help us to better understand the definition.

**Definition 5.** The level set \( L_F(d) \), of a scalar field \( F = (f, D) \), is the set of points in \( D \) of equal
scalar value \( d \). More formally, \( L_F(d) = \{ p | p \in D \land f(p) = d \} \).

For computation of the contour tree we use an algorithm by Carr et al. [11]. As input, the
algorithm expects a scalar field \( F^\ast = (f^\ast, M) \), where \( M \) is a mesh that is a simplicial complex
(Definition 3) and \( f^\ast(v) \) defines the scalar value for each vertex \( v \in M \). For the algorithm to
produce correct results (i.e., topological changes in the contour tree are recorded at the scalar
value and position at which they actually occur), it assumes that all critical points (Definition 6)
are vertices of \( M \). And as a result of this, assumes the scalar field \( F^\ast \) can be extended over the
entire plane, by means of piecewise-linear interpolation over the simplices of \( M \).

As input for \( M \) we use the Delaunay triangulation of \( P \). The scalar value at vertices of \( M \) are
directly taken from the density map \( F = (f, \mathbb{R}^2) \) (as computed in Section 4.3.1), so \( f^\ast(v) = f(v) \)
for all \( v \in M \). In other words, \( F^\ast \) is essentially based on \( F \), and uses linear interpolation over \( M \)
to interpolate the scalar values that originally were already defined in \( F \). Therefore, we say \( F^\ast \) is
an approximation to the actual density map \( F \).

**Definition 6.** Critical points in scalar field \( F = (f, D) \) are points in \( D \) at which topological events
occur. These are local maxima (new contours appear), local minima (contours collapse) and saddle
points (contours split or merge). All other points are called regular points. A more detailed
description is given by Bajaj et al. [4].

The notion of a level set (Definition 5) is now well understood in terms of the scalar field \( F^\ast \). For
a level set \( L_{F^\ast}(d) \), we call each connected component in the mesh \( M \) a contour.

**Definition 7.** We use the definition of a contour tree as given by Carr et al. [11]. The contour
tree of the scalar field \( F^\ast = (f^\ast, M) \) is a graph structure in which:

1. Each leaf vertex represents the creation or collapsing of a contour at a local maximum or
   minimum in \( F^\ast \).
2. Each interior vertex represents the joining and/or splitting of two or more contours at a
critical point in \( F^\ast \).
3. Each edge represents a contour in \( F^\ast \) for each scalar value between the values at the end-
   points of the edge.

The algorithm itself is divided into three stages: (1) sort the vertices of \( M \) according to their
scalar value, (2) compute the join- and split tree, and finally, (3) combine these into a contour
tree. The join tree describes creation and merges of contours, and the split tree the collapsing and
splitting. Both are computed by vertically sweeping the density map, over the sorted vertices of
\( M \), and recording the topological events during this sweep. (The procedures to compute both are
actually identical: for the join tree the sweep is done top-down and for the split tree bottom-up.)

For further details, and a discussion on the time complexity and construction of a contour tree we
refer to the original work [11, 33].

**Modifications.** In the following subsections we describe some modifications and additions that
are required by our other algorithms that use the contour tree. First of all, it must be noted that
there are no split vertices in our data. This is because in a density map \( F = (f, \mathbb{R}^2) \), at a local
minimum, contours only merge, and it is not possible that a contour is created. The reason for this is that the scalar function \( f \) is explicitly defined, i.e., each point \( p \in \mathbb{R}^2 \) only has a single scalar value \( f(p) \). In practice, this means that we only have to compute the join tree, because it is identical to the contour tree.

### 4.4.1 Critical point insertion

In the introduction to contour trees we state that the algorithm requires all critical points to be located at the vertices of \( P \). Using the density map \( \mathcal{F} = (f, \mathbb{R}^2) \), and the Delaunay triangulation \( \mathcal{M} \) as input, this is however not necessarily the case. We demonstrate this in 1D for the local minimum (Fig. 4.6(a)) and the local maximum (Fig. 4.6(b)). In 2D the local minimum case with two Gaussians will actually be a saddle point and not a local minimum.

Without modifications, the topology described by the contour tree \( \mathcal{T} \) over \( \mathcal{F}^* \) would therefore have inconsistencies with the actual topology of the density map \( \mathcal{F} \). This causes problems when we use \( \mathcal{F} \) and \( \mathcal{T} \) in conjunction, for example, during the rendering of the regions (Section 5.3). To solve this we could simply use \( \mathcal{F}^* \) instead of \( \mathcal{F} \). However, due to the piecewise-linear interpolation in \( \mathcal{F}^* \) this would create less smooth region shapes.

**Saddle points.** The first problem is caused by the algorithm detecting saddle points earlier in \( \mathcal{F}^* \) than they actually occur in \( \mathcal{F} \). This means that it can happen that in the contour tree \( \mathcal{T} \) a set of points will be considered a single contour, while visually, they are still two separate contours.

Note that local maxima and local minima are less of a problem. The first represents a contour appearing, and the second a contour collapsing (Definition 6). In our case, the latter is always a hole in a region that closes. Because we only render regions that occur exactly at densities of points in \( P \) (see Section 4.5), missing a local minimum or maximum does not cause any problems when rendering.

**Bounding polygon.** The second problem has less to do with the topology of \( \mathcal{F} \) itself, but rather with some subtleties during the rendering of the regions. For rendering we require the bounding polygon of a region \( R \) in \( \mathcal{F} \) to be available. This bounding polygon is computed by looking at neighbors in the Delaunay triangulation of points in \( R \), and deciding whether or not these neighbors are still in \( R \) (depending on their scalar values in \( \mathcal{F} \)). For more details on this see Section 5.4.1.

However, the bounding polygon that is computed might not really bound \( R \), as illustrated in Fig. 4.7. Here \( R \) crosses edge \( e \) (the thick edges represent the computed bounding polygon), which is undetected during computation of the bounding polygon. The reason that this happens is because we only look at scalar values at the vertices, and not over the edges themselves. We could solve this after computing the contour tree, e.g., during the actual computation of the bounding polygon, but fixing it in advance will also benefit computation of the area approximation (Section 4.4.2).

We solve both the first and the second problem by looking at each edge \( e \) in the triangulation. If this edge contains a point \( q \) with a lower scalar value \( f(q) \) than both the endpoints of \( e \) (we call this point \( q \) a min-point), or if \( q \) has higher scalar value \( f(q) \) (max-point), we add this point to the set \( P^* \). If all edges are done, we recompute the Delaunay triangulation over the points \( P \cup P^* \), and use this as the input for the contour tree instead.

**Alternatives.** Detecting all saddle points can also be done globally in \( \mathcal{F} \), for example, using some kind of hill climbing procedure as used in the Denclue 2.0 algorithm [22]. However, it suffices to
do this only over edges of $\mathcal{M}$, because we do not require the exact position of the saddle points, but only the exact densities over an edge where two contours merge.

### 4.4.2 Augmented contour tree

The actual contour tree $\mathcal{T}$ only contains nodes for the critical points in $P$ (called super-nodes). These super-nodes represent contracted contours. All other regular points are contracted into the first super-node down the branch. However, we want the contour tree to contain nodes for all these regular points as well, i.e., all points that get included in a contour but do not cause splits or merges. This means the contour tree does not only contain all contracted contours, but all possible contours for scalar values occurring at any of the points in the input $P$.

We require this, because it enables us to select and render contours for any point in $P$, instead of just the contracted contours only. This variant $\mathcal{T}^*$, of the contour tree $\mathcal{T}$, is called the augmented contour tree. The regular points are stored in the contour tree as a (single) child nodes of the lowest density node of the contour (Fig. 4.8). Note that the augmented contour tree has some similarities with the dendrogram computed by the hierarchical clustering methods described in Section 4.2: given a node $n$, the subtree starting at $n$ represents all the points contained in the contour of $n$. In the original algorithm, the regular nodes are explicitly contracted into the super-nodes, so if we require the augmented version of the contour we can simply omit this contraction step.

To support our other algorithms that make use of the contour tree, we have to precompute certain ancillary information, and store this in the nodes itself. We do this by using a single bottom-up traversal of the tree, computing the required information as we go. In the remaining parts of this section we list, and briefly describe, the ancillary properties that we compute, and why we need them. To ease the discussion we first introduce some notation.

Recall that we consider every node $n$ in the contour tree $\mathcal{T}$ to represent a possible region $R$ that we can display on the map. This region is represented by the filled contour at density $d(n)$ covering the subtree of $\mathcal{T}$ with root $n$ (i.e., $n$ and its children).

**Definition 8.** We say a triangle $t$, is fully contained in the region $R$ of node $n$, if all three vertices of $t$ are contained in the subtree with root $n$ (i.e., all vertices are covered by $R$). A triangle $t$ is partly contained in $R$ if it has either one or two vertices of $t$ contained in $R$.

![Augmented contour tree](Figure 4.8: Augmented contour tree.)

![Contour at density d(n) covering the subtree with root n.](Figure 4.9: R is represented by the contour at density d(n) covering the subtree with root n.)
Approximated area. We store an approximation of the total area of each region $R$ in the node $n$. This can be used to either display statistics for $R$, or during selection of regions, as a measure of importance of $R$. We distinguish three types of approximations, all defined in terms of the sum of the areas of a set of individual triangles:

1. All triangles in $\mathcal{T}_D$ partially contained in $R$ (Fig. 4.10(a)).
2. All triangles in $\mathcal{T}_D$ fully contained in $R$ (Fig. 4.10(b)).
3. All triangles in $\mathcal{T}_D$ fully contained in the region $R$, and the triangles inside $R$ formed by interpolation within triangles in $\mathcal{T}_D$ partially contained in $R$ (Fig. 4.10(c)).

Of these three methods, the third gives the best approximations. The total error (difference between real area and approximated area) is smaller, because at triangles on the boundary of the region the interpolated area is computed, instead of simply omitting, or fully adding the area of the triangle. However, the drawback is that this method is computationally more expensive than the first two methods. The reason for this is that we have to re-interpolate along the entire boundary of each region individually.

Consider for example that we are processing a fresh vertex $v \in \mathcal{M}$. For the sake of argument we assume $v$ to be a regular vertex (no split or merge vertex). If we add $v$ to the contour tree, we have to compute its approximated area. By definition of fully inside and partially inside, for method 1 and 2 we only have to look at the triangles that touch $v$. However, for the third method, we have to look at all triangles on the boundary of the region of $v$. This is because we cannot use the interpolated value we have previously computed for the parent of $v$. The density of $v$ will be less, and thus at all border triangles, the interpolated area will change.

By default, we use the second method. The first method is more likely to produce approximations with significant error for datasets with sparse areas. This is because such datasets typically will result in large triangles in the triangulation (illustrated in Fig. 4.11), and therefore possibly result in high error in the approximation for the first method. Note that for the second method this is less of a problem, because typically these sparser areas indicate that regions end there, and therefore the right choice is not to include the large triangle.

Note that we could also use other methods to approximate the area of a region, e.g., count the number of grid cells in the density map (see Section 5.2.1). This is a less complex structure than the triangulation, and would therefore simplify computations. However, this would give high error if the resolution of the input set of points is higher than the resolution of the grid.
**Number of points.** Store the total number and total weight of all points that are contained in the region $R$ of $n$. This can be used for purposes such as displaying statistics about $R$, or coloring $R$ based on the number of contained points (i.e., indicating importance).

**Densities.** We keep track of minimum, maximum and mean densities, and the points at which they occur. This information can be used to display statistics about a region, or for example, to determine the color of the region, as well as give candidate points that we can use to display symbols. For example, if we assume the densest point to be a central point in the region, it would be a good candidate to position the label of this region at.

**Bounding box.** We keep track of the bounding box of the points contained in the region of $n$, which is used for the positioning of region symbols.

### 4.5 Selecting regions

Our primary use of the contour tree is to select regions that we want to show on the map. Besides this, we use it to store ancillary information (e.g., weighted point count, approximated area) of these regions. We can traverse the contour tree, and depending on the level of detail that we want to show, make a local decision per region whether or not to show it. This means that we have direct control over what will be visible on the map. On the other hand, we can also indirectly influence this by changing properties of the kernel we use during computation of the density map (Section 4.3). For example, more smoothing means important regions will become more dominant on the map, while less smoothing will result in a lot of smaller regions. In this section we introduce three methods that we use for the selection of regions from the contour tree. We call these region selection strategies.

![Figure 4.12: Selection of regions in a contour tree $T$ (a) can intuitively be seen as the intersection of the height field with a plane (b).](image)

The first is essentially a standard contouring approach: pick a density $d$, and display all contours for this density. For the second strategy, we use the contour tree to compute a selection that covers a certain percentage of the total map area, we call this the map coverage strategy. The third method turns absolute density into relative density, with the aim to uncover the local maxima. In the remaining part of this section we describe these three selection strategies in more detail.

**Comparison.** All three strategies produce different maps, which we cannot directly compare based on their input parameters. Therefore, we introduce a single control parameter: the percentage of map coverage. We create mappings between this control parameter and original parameters of the three strategies. This way we can accurately control the percentage of map covered by a map generated for each of the three strategies. We can use this control parameter as a basis to compare the different maps.
4.5. SELECTING REGIONS

4.5.1 Constant Density

The standard way of rendering contours in a contour map is by rendering each contour at given density \( d \). For the constant density strategy we mimic this by selecting all regions that have this density \( d \), i.e., we select a single level from the contour tree \( T \). However, the density of a region is usually not exactly equal to \( d \). Therefore, we traverse down \( T \), and select the first nodes (representing regions) we encounter that have a density lower than or equal to \( d \). This means that there is a possible difference between \( d \) and the actual densities of the regions that are selected. We consider this distance neglectable, because in practice we assume large sets of points, which means individual points will lie close to each other and as a result the difference in density between connected nodes in \( T \) is very small.

Mapping to percentage coverage. We create a mapping between percentages of map covered \( C \) and the densities \( d \) to be used to obtain this coverage. The mapping is a simple list, where the key of each record (we call a mapping slot) represents a percentage and the corresponding value a density. The size of this list determines the accuracy we can achieve. By default we use 1000 mapping slots, which means that for this strategy, the percentage coverage control parameters is accurate up to one tenth of a percent. For example, if we want to know the density that achieves a 12.5% coverage of the map, we look at the density stored at the record with key 125.

First we sort all regions in the contour tree according to their density in descending order. So the first item on this list represents the contour that has the highest density (e.g., lives longest), and the last item the contour with the lowest density. Only showing the first item results in the lowest possible map coverage \( C \) while still having a visible region (only this single region). As we move up the list, \( C \) will gradually increase, because each new item we see represents the creation of a new region, or the expansion of an existing region (points get included).

The mapping is created by doing just this, iterating over the list, updating the coverage \( C \). If we are about to exceed a coverage \( c \) (i.e., \( C \leq c \) and for the next item \( C > c \)), for example \( c = 0.1\% \), we fill the slot corresponding to 0.1% with the current density, and move on to the next slot (0.2%). This is repeated until all slots are filled. Later, if we need to cover for example 20% of the map, we can use the mapping to find the density \( d \) that achieves such a discrete covering using this selection strategy.

4.5.2 Map Coverage

This strategy is based on the conceptual idea that we can achieve a more or less uniform distribution of regions over the map if we subdivide the map into equal sized squares (illustrated in Fig. 4.13), and enforce that each square is covered by an equal amount of region area. The smaller the squares, the more uniform the distribution will be. However, to simplify computation, we do not use an actual subdivision into squares, but a more spatial subdivision, that follows the hierarchy of the contour tree \( T \).

The algorithm requires as input a threshold percentage area \( B \), we call this the global budget. The goal is to find a subset \( S \) of non-overlapping regions that have a total area close to, but not exceeding, the global budget \( B \) (the upper bound is defined more precisely later on). We do a top-down traversal, starting at the root of \( T \), and spread the budget \( B \) evenly across the branches of \( T \) (Figure 4.14 illustrates this). Each branch gets assigned a local budget \( b \) that is propagated through the tree, initially, for the root node \( b = B \). During the traversal, at a node \( n \), there are two possible cases:

1. The region \( R \) represented by \( n \) has a total area not exceeding the local budget \( b \). This means that we can safely display \( R \), and thus we add it to \( S \), and stop traversing the current branch.
2. Otherwise, we need to propagate $b$ to the children $C$ of $n$. We do this proportionally to the area of the children. For a child $c \in C$, the new budget $b_c$ is defined as

$$b_c = b \frac{\text{area}(c)}{\sum_{c' \in C} \text{area}(c')}$$

(4.12)

where $\text{area}(m)$ denotes the approximated area of the region represented by $m \in T$.

![Figure 4.14](image)

*Figure 4.14: Illustration of the propagation of budget $B$ over the contour tree $T$."

The difference between the constant density and the map coverage strategies is illustrated in Figure 4.15.

**Notes on the upper bound.** This is a greedy approach; at every split in the contour tree the budget is divided over the children according to only properties of the children (i.e., a local decision). In total, the assigned budget is always equal to the budget that was assigned to the parent (this follows directly from Eq. (4.12)). The most precise upper bound on the total area of the resulting selection we can give is: for each branch the area is maximized and does not exceed its assigned budget area $b$. Therefore, the budget $B$ is never exceeded. It does mean that we can possibly find a resulting set $R$ that has an area closer to $B$, if we violate the budget in one of the branches deliberately. However, we do not do this as in practice the resulting area of $R$ is very close and we consider the difference neglectable.

![Figure 4.15](image)

*Figure 4.15: Maps of population distribution of the Netherlands (2 km kernel bandwidth and 2.5% map coverage) showing difference between (a) constant density strategy, and (b) map coverage user a spread factor of 1.0 (b).*
4.5. SELECTING REGIONS

**Total area.** We express the budget $B$ as a percentage of the original size of the map. This means that we can satisfy requests such as cover 20% of the map with regions. However, it is important that this percentage behaves consistently for different types of maps. For example, if we take the bounding box of the input set of points $P$ to be the total map area, there would be a large visual difference between maps that contain a lot of background area (e.g., maps containing large parts of an ocean) and maps that contain no such background area (e.g., city centers).

**Definition 9.** We consider the total area $A$ of the map to be all non-background area. More formally, we define this as the union of the area of all non-null contours in the contour tree $\mathcal{T}$:

$$A = \text{area}\left(\bigcap_{n \in \mathcal{T}} C(n)\right),$$

where $C(n)$ denotes the contour of node $n \in \mathcal{T}$.

**Spread.** The map coverage algorithm tries to fill the map with regions up to a certain threshold area as fairly as possible. As we decrease the kernel bandwidth, this might give an undesired result. A small bandwidth results in less smoothed data and we could end up with the map being fully covered with small regions (Fig. 4.16(b)). We introduce an extra parameter, the spread factor, that we can use to influence the spread of the regions. In Figure 4.16 we illustrate the effect of this parameter.

![Figure 4.16: Illustration of the effect of the spread parameter, using a kernel bandwidth of 2 km and 5% map coverage: (a) spread = 0, (b) spread = 0.33, (c) spread = 0.66, and (d) spread = 1.](image)

Recall, when distributing the budget $b$ in a node $n$ over its children $C$, we do this proportionally to the area of the children. We introduce a spread parameter $s$ into this scheme, by making the division proportional to the importance of the regions as well. By default, we use the weighted number of points contained in a region (denoted by $\text{sizew}$) as the importance. We incorporate this into the propagated budget $b_c$ as defined in Equation (4.12) as follows:
By varying the spread parameter $s$, we can change the influence of the weighted point count on the ratio used to spread the budget over the children $C$. Setting $s = 1$ means the weighted point count will have no influence (the equation simplifies to Eq. (4.12)). We consider this to be the maximal spread. Taking $s = 0$, minimal spread, means we multiply the original ratio with $\sum_{c' \in C} \text{size}(c') (1 - s)$, and thus assign proportionally more weight to region with a higher weighted point count.

**Nesting.** The map coverage algorithm graciously extends to allow for nested regions. For the original algorithm we consider the union of all non-zero density contours to be the map region. We take the total map area to be the area of this region. The function of the map coverage algorithm is to cover a certain percentage of this region with sub-regions, so essentially, our regions are already nested regions of the map region itself. Because of the naturally hierarchical structure of spatial data, we can simply generalize the map coverage algorithm such that it does not only apply to the map region itself, but to any arbitrary subregion as well. We introduce a new parameter called the nesting threshold, which indicates a minimum area threshold for the regions we want to recursively apply the coverage algorithm to (Fig. 4.17). In other words, we add nested regions to any region that meets a minimum size requirement. Adding nested regions to all regions would make the map more cluttered and less readable.

![Figure 4.17: Examples of maps showing nested regions, using a kernel bandwidth of 2 km and 10% map coverage: (a) using the constant density strategy (selecting levels at 8%, 30% and 80% density), and (b) using the map coverage strategy (20% recursive coverage, 0.5% recursive threshold and spread = 0.5).](image)

### 4.5.3 Relative Density

The function of the relative density method is to uncover local maxima by turning absolute density into relative density. This is a two-step process, in which we use only operations on the density maps to influence the selection of regions. The actual region selection is done in the same way as for the constant density strategy, i.e., select a single level of the contour tree.

Given input $P$, we first compute a density map $F_1 = (f_1, R^2)$ using a kernel bandwidth $h_1$ (e.g., Fig. 4.18(a)). In the second step, we compute a density map $F_2 = (f_2, R^2)$ using a significantly larger kernel bandwidth $h_2$. We consider this to be the average density (Fig. 4.18(b)). These two are combined by dividing the first by the second density map, resulting in the scalar field $F_{res}$ depicting relative density (Fig. 4.18(c)). More formally, $F_{res} = F_1 / F_2 = (f_{res}, R^2)$, where
4.5. SELECTING REGIONS

Figure 4.18: Plots of the scalar fields $F_1$, (a) using a kernel bandwidth of 6 km, $F_2$ (b) using a kernel bandwidth of 20 km, and the resulting scalar field $F_{\text{res}}$ (c) depicting relative density.

$$f_{\text{res}}(p) = \begin{cases} f_1(p) & \text{if } f_2(p) > 0 \\ 0 & \text{otherwise} \end{cases},$$  \hspace{1cm} (4.15)

for all points $p \in \mathbb{R}^2$. Because we assume $h_2$ to be larger than $h_1$, and all values in the density map to be positive, we know that if $f_2(p) = 0$ at any point $p \in \mathbb{R}^2$, then $f_1(p) = 0$ as well (because of the smaller kernel bandwidth). Therefore, in this case, it makes sense to use 0 for the resulting scalar field $F_{\text{res}}$.

**Mapping to percentage coverage.** For this strategy we only change the scalar field, but still use the same selection method as for the constant density strategy. This means that the mapping to the percentage coverage control parameter is identical to what is described for the constant density strategy.

**Boundary preference.** A drawback of this method is that it suffers from boundary preference. We reason that the density map with a significantly larger bandwidth, in essence, represents the average bandwidth at a given location. This is unfair in areas close to the boundary of the dataset. For example, at natural boundaries such as coastlines, in most datasets there will be no points in areas covered by water. This means the average density will be lower, and therefore, regions at these boundaries will be considered local maxima quicker than interior regions (Fig. 4.19).

Figure 4.19: Brabant using a 2 km kernel and 10% map coverage: (a) absolute density and (b) relative density using a second density map with a 20 km kernel.

We could think of techniques to reduce this, for example, by adding extra artificial density to the boundary of the dataset. However, in our case, we consider this problem to be just a minor drawback, because it only happens at the boundary, and we can still compare maps between the different strategies purely based on their interior.
We turn the model, as discussed in previous chapter, into a working prototype. In this chapter we describe the various implementation details specific to the different components of the model as well as the prototype itself. Implementation is done in C++, in combination with the QT library and OpenGL to produce the actual graphics.

5.1 Input

We use an input format for storage and loading of datasets (Fig. 5.1), which is in plain text ASCII, and kept simple by design. The structure of the format is flexible and allows us to easily convert or import data from other plain text data sources. We expect most of the data to be either available in plain text, or easily convertible to plain text using existing tools. In this section we describe the expected input data and also possible preprocessing steps that might be needed to make it suitable as input for our model.

**Input format.** This intermediate format contains a small header which specifies the number of points and the input coordinate system (WGS84 or Cartesian). Additionally, the bounding box of the points is given; this is used to scale and translate the points to fit exactly within the viewing area. The header is followed by a list of the actual points. For each point the $x$- and $y$-coordinates and the weight are stored.

**ESRI grid format.** The ESRI grid format is a popular format for the exchange of spatial datasets, used by the ArcInfo family of GIS applications [30]. A large collection of geographic data is available on the Internet in this format. To be able to directly use these datasets, we have implemented a small special purpose application for the conversion from the ESRI grid format to our input format described above.

**Map Projection.** In geographic space, any position on the Earth is uniquely addressable using a spherical coordinate pair $(\lambda, \phi)$, where $\lambda$ is the longitude and $\phi$ is the latitude. The longitude is the angle from West to East and the latitude is the angle from North to South. Directly plotting the points using these spherical coordinates would lead to much distortion, therefore, we need to project the points to the plane. This process is known as map projection.
Map projections exist in a wide variety, where in general three different classes are identified: **cylindrical**, **conic** and **planar** [20, 41]. Important map attributes, such as area and shape of features, and distances between features on the map are considered to be important. Depending on the requirements for the projection, preservation of one (or more) of these properties might be required.

A map projection is said to be: **equal-area**, if it depicts the area of all regions in correct proportion, **conformal**, if all shapes and angles of small features are preserved at every location, and **equidistant**, if it preserves distances from a single point to any other point on the globe. For a complete list of these properties we refer to Feeman [20].

We use a common projection called the Mercator projection. This is a conformal cylindrical projection, which maps each circle of longitude (line with constant latitude) to horizontal lines in the plane, and the circles of latitude (line with constant longitude) to vertical lines. This projection places the $x$-axis at the equator of the earth and the $y$-axis at any arbitrary longitude $\lambda_0$. Any point $p$ with the spherical coordinates $(\lambda, \phi)$ can be projected to the Cartesian coordinates $(x, y)$ [51] using

\[
x = \lambda - \lambda_0; \\
y = \tanh^{-1} (\sin (\phi)).
\]  

(5.1)

Note that in the remaining part of this chapter we generally use $(\lambda, \phi)$ to refer to a point in spherical coordinates (longitude, latitude) and $(x, y)$ to refer to a point with Cartesian coordinates $x$ and $y$.

**Geographic distance.** Due to irregularities of the Earth’s surface (e.g., hills), it is not feasible to try to calculate the exact geographic distance between two points. A common abstraction is to assume the Earth to be a perfect sphere (or sometimes a perfect ellipsoid) and calculate the distance as the shortest distance between two points on a sphere (measured along its surface). This is called the **great-circle distance** (Fig. 5.3). This abstraction simplifies calculations as we can assume the radius $R$ of the Earth to be constant, or near constant. The great-circle distance $\text{distance}(p_0, p_1)$ [20] between two points $p_0 = (\lambda_0, \phi_0)$ and $p_1 = (\lambda_1, \phi_1)$ is given by

\[
\text{distance}(p_0, p_1) = R \cdot \cos^{-1} (\cos \phi_0 \cdot \cos \phi_1 \cdot \cos (\lambda_0 - \lambda_1) + \sin \phi_0 \cdot \sin \phi_1).
\]  

(5.2)

However, we use an approximation that is not as accurate, but greatly simplifies our derivations (Section 5.2.1), based on a spherical Earth projected to the plane

\[
\text{distance}(p_0, p_1) = R \cdot \sqrt{(\lambda_0 - \lambda_1)^2 \cdot \cos (\phi_m)^2 + (\phi_0 - \phi_1)^2},
\]  

(5.3)

where $\phi_m$ denotes the mean latitude given by $\phi_m = (\phi_0 + \phi_1)/2$.

A standard simplified model is the World Geodesic System (WGS84). Since most of our sample datasets use this model, we adopt it as well. WGS84 assumes the Earth to be a perfect sphere, using a constant radius $R$ of 6.378.137 m.

**Merging duplicates.** Because some of the algorithms we use assume all points in the input dataset to have a unique position (no duplicates), we have to make sure the input dataset does not contain any of these. We add an extra preprocessing step that merges any pair of duplicate points into
a single point by using the sum of their weights. This is repeated until the dataset no longer contains any duplicates. After merging the duplicates, the density map computed using KDE (Section 4.3.1) is identical to the density map if we would have computed it without merging the duplicates.

**Bounding box.** To eliminate the need to specifically handle certain special cases of some of the algorithms, we add a bounding box that encloses all points in the input \( P \). To make sure the vertices of this bounding box do not intervene with the algorithms we set their weight to 0.

**Generating artificial datasets.** Testing certain properties of algorithms that we use can be done using real datasets, however, it is also useful to be able to quickly generate artificial datasets that contain certain features that we want to test.

We have implemented a basic random method that generates an artificial dataset \( D \) based on any regular image \( I \) (Fig. 5.4). The image is first converted to gray-scale, where we consider the lightness of a pixel \( i \) (in the range of 0...1) to denote the probability \( pr_i \) that \( D \) contains a point at the position of \( i \). We then simply generate the dataset by iterating over all pixels, and for a pixel \( i \), draw a point with probability \( pr_i \). Finally, we transform from pixel space to unit space by scaling the coordinate pair with \( \frac{1}{\max(w,h)} \), where \( w \) and \( h \) denote the width and height of \( I \) respectively.

Reducing or increasing the amount of points in \( D \) can be done by scaling all probabilities \( pr_i \) (for each pixel \( i \in I \)) with a scale factor \( p_{sf} \). This scale factor is configurable from the user interface. By multiplication of the probability with a sigmoid shaped function (Fig. 5.5) we can influence the sharpness of the dataset. This function is configurable from within the user interface, by linear interpolation between the sigmoid function \( S(x) = \frac{1}{1 - e^{-(x+12)/6}} \). Other transformations such as mirroring, translation, and rotation within the coordinate system are possible as well. Using this method we can quickly generate artificial datasets and have a lot of control over the features contained in these datasets.

**Sampled data.** Because we use Kernel Density Estimation to smooth the point data (Section 4.3.1) we do not need any special treatment for data that is sampled in a grid.

### 5.2 Density maps

In the previous chapters we only used a single density map. However, in our prototype we have the ability to compute up to two density maps. Using a combination operator these can be combined into a single density map that is used for the actual visualization (shown in Fig. 5.6).
5.2. DENSITY MAPS

From the input set of points \( P \) we can compute a first density map \( F_1 = (f_1, \mathbb{R}^2) \) and optionally a second density map \( F_2 = (f_2, \mathbb{R}^2) \). After computation, we can apply some post processing to the density map using a post operator. Currently we only have the \( \log \) operator defined that allows us to apply logarithmic scaling to the density map. The effect of this is illustrated Section 3.1 (Fig. 3.2).

We define the \( \log \) operator \( f_{\log} \) on a density map \( F = (f, \mathbb{R}^2) \) as

\[
 f_{\log}(p) = \log_e(1 + f(p)). \tag{5.4}
\]

Note that we apply the \( \log \) function to the sum of 1 and \( f(p) \), this is to make sure the scaled value \( f_{\log}(p) \) does not become negative.

**Combining the two density maps.** The combination operator \( \oplus \) defines how the two density maps \( F_1 \) and \( F_2 \) are to be combined into the resulting scalar field \( F_{res} \). Formally \( F_{res} = F_1 \oplus F_2 = (f_1 \oplus f_2, \mathbb{R}^2) \). Currently we only have the \( \text{div} \) operator defined, which essentially divides the first density map with the second density map. However, this setup allows us to extend the model with extra combination operators in the future.

Besides having the ability to compose a single density map out of two, we can also add layers to the map based on either one of the two density maps. For example, we can use a small kernel density map to generate the regions and add a layer hinting average density by applying hill-shading to the second density map (Section 5.5.3), computed using a larger kernel (Fig. 5.7).

![Figure 5.7: Example of combining two visualizations based on different density maps: detailed regions (using a 1 km kernel) and hill-shading (using a 5 km kernel).](image)

In the remaining part of this section, we describe the technical details on how we compute a density map.

5.2.1 Computation

Having an efficient method to compute density maps has a number of advantages. On the one hand it allows our model to cope with larger amounts of points, but more importantly, we can use a detailed grid which in turn results in a visualization of higher graphical quality.

Conceptually, we define the density map as the scalar field \( F = (f, \mathbb{R}^2) \). This means that we consider a density estimate \( f(p) \) to be available for each point \( p \in \mathbb{R}^2 \). Computationally this is possible if we simply estimate the density at this \( p \) using KDE. However, in practice this is not very efficient, because this means we have to look at all observation points in the input \( P \) to get a single estimate \( f(p) \). Instead, we use a grid, and precompute the density estimate for each grid cell. (The grid centers are used as evaluation points.) We then use bilinear interpolation between these grid centers to get a density estimation for any arbitrary point \( q \in \mathbb{R}^2 \). We generally refer to this grid, containing density estimates, as being the density map.
Splatting. Computation of the density map is done by splatting the points (in computer graphics terminology). This is illustrated in Figure 5.8. We define a grid $G$ of $m \times m$ cells, where we call $m$ the size of the grid. For each observation point $p$ in the input $P$, we add a contribution $c$ to all grid cells of $G$. This contribution depends on the weight of $p$, the kernel function, the kernel bandwidth used (see Section 4.3.1 for a more detailed explanation of Kernel Density Estimation), and the distance of the grid cell center (the evaluation point) to $p$. Note that we do not have to iterate over the entire grid, but only the portion in the vicinity of $p$ that is within the radius of the kernel. This is because we explicitly choose a kernel function that has no contribution outside of its radius.

Notes on the time complexity. The time complexity of this algorithm depends on the size of the input $n$ and the size of the grid $m$. We compute a KDE estimate for each of the cells in $G$, which gives us a time complexity of $O(nm^2)$.

Implementation on graphics hardware. We implement the splatting procedure on graphics hardware, using OpenGL’s vertex buffer objects (VBOs), additive blending, and float textures to directly store the density map on the GPU’s memory. In the rest of this chapter, when we refer to the density map, we generally refer to the density map stored in the texture.

We first compute the density map $\mathcal{F}_G = (f_G, G)$ in geographic space (Fig. 5.9). For this we use OpenGL’s fragment shader. Essentially, this allows us to write a small program that is executed as part of the OpenGL graphics pipeline for each individual pixel that we draw. The texture used to store the density map has the same dimension as our grid (i.e., each pixel represents a single grid cell).

For each observation point $p \in P$, we invoke the fragment shader by drawing a quad - centered at $p$ - that approximately bounds the kernel. In the fragment shader itself, we compute the distance $d$...
between the current pixel (or grid cell) and the observation point \( p \), using the geographic distance function as defined in Equation (5.3). Based on this distance \( d \) and the kernel function \( K \), we compute the contribution of \( p \) to the grid cell, and store it in the texture. Thanks to OpenGL’s additive blending, the end result - after splatting all observation points - is a grid in which each grid cell contains the sum of the contributions of all observation points to this grid cell.

The width \( \Delta x \) and height \( \Delta y \) of this quad have to be specifically computed for each individual observation point, because of the distortion along the \( y \)-axis (Fig. 5.10). We use the \( \Delta x \) and \( \Delta y \) notation to avoid confusion with the \( h \) we commonly use to refer to the kernel bandwidth. We use the distance function as defined in Equation (5.3) to derive equations that approximate both the width and height in radians, which equal the given geographic distance \( h \) in meters (the kernel bandwidth). Given a grid cell \( p \), centered at the geographic position \( (\lambda, \phi) \), and the kernel bandwidth \( h \) in meters, we can find the required width \( \Delta x \) by finding a point \( q = (\lambda^*, \phi) \) with distance \( h \) to point \( p \). In other words, we locate a point \( q \) that is at geographic distance \( h \) of point \( p \) and is on the same latitude. We then use the unprojected distance (\(|\lambda - \lambda^*|\)) as the width \( \Delta x \). We find \( q \) by solving

\[
h = \text{distance}(p, q)
= R \cdot \sqrt{(\lambda - \lambda^*)^2 \cdot \cos^2((\phi + \phi^*)/2) + (\phi - \phi^*)^2}
= R \cdot \sqrt{(\lambda - \lambda^*)^2 \cdot \cos^2 \phi}
= R \cdot |\lambda - \lambda^*| \cdot |\cos \phi|, \text{ hence}
\Delta x = |\lambda - \lambda^*| = \frac{h}{R \cdot |\cos \phi|}.
\] (5.5)

We do the same for \( \Delta y \) by fixing the longitude \( (q = (\lambda, \phi^*)) \) and solve

\[
h = \text{distance}(p, q)
= R \cdot \sqrt{(\lambda - \lambda^*)^2 \cdot \cos^2((\phi + \phi^*)/2) + (\phi - \phi^*)^2}
= R \cdot \sqrt{(\phi - \phi^*)^2}
= R \cdot |\phi - \phi^*|, \text{ hence}
\Delta y = |\phi - \phi^*| = \frac{h}{R}.
\] (5.6)

**Projection to Mercator.** Finally, we project this density map to a Mercator projection. To store this projected density map \( \mathcal{F}_M = (f_M, G) \) we create a texture of the same dimension as our grid. The fragment shader is invoked by drawing a quad of the size of the texture. In the fragment shader we map the center of each pixel from screen space to Mercator space to get the point \( p = (x, y) \). We use the inverse Mercator projection to project this point to its geographic location \( q = (\lambda, \phi) \) [51] using equations

\[
\lambda = x + \lambda_0;
\phi = \tan^{-1}(\sinh(y)).
\] (5.7)

The last step is to proportionally scale \( q \) to fit within the bounding box of \( \mathcal{F}_G \), and sample the value of this grid cell from \( \mathcal{F}_G \).
Small resolution density maps. Because computational models on graphics hardware are optimi-
mized to be fast, but not necessarily as accurate as on the CPU, it is possible that numerical
errors have a negative impact on the visualization (i.e., jagged edges). This happens if we want to
compute density maps for relatively small geographic regions. Due to numerical errors with the
backwards Mercator projection we get a projected density map $F_M$ that is less smooth than the
unprojected version $F_G$. This means that regions and region borders in the visualization will look
jagged.

However, in a Mercator projection the actual distortion for very small regions is minimal. Up to a
certain threshold $\epsilon$ (in degrees), we consider this variation in distortion to be neglectable. For any
region smaller than $\epsilon$, we compute the smallest and largest distortion (latitude closest and farthest
away from the equator respectively), and instead of applying the backward Mercator projection,
we scale linearly between the smallest and largest distortion. For $\epsilon$ we have chosen a value of one
arc second (approximately 2 km$^2$).

Grid-size. By default, we use a grid equal to the largest square we can fit into the projection area.
In other words, each grid cell is exactly the size of a pixel. This means when used directly for
display on screen, our density map contains the maximum amount of detail possible. Reducing
the size of the grid would mean the quality of the image would go down, since we use multiple
pixels to represent the same grid cell. We could also increase the size of the grid, this means the
density map will contain more information than we can actually display. However, it might still
be useful because whenever we need the gradient (e.g., when drawing the boundaries or shading)
the approximation of the derivative gets better as the grid-size increases.

Computing the minimal and maximal densities. To compute the minimal and maximal values in
the density map on the GPU, we use a common GPU programming technique called ping-ponging
between two textures (source and destination). Initially the density map is copied to the source
texture, and for each pixel its maximum value is determined among the neighboring pixels to
the right, bottom and bottom-right) and copied to the destination texture. This results in the
destination texture being one fourth of the size of the source texture. The source and destination
textures are swapped (the ping-pong step), and the entire process is repeated, until the destination
texture only exists of a single pixel. The maximal value is now stored in this pixel.

In theory, this is actually slower than sequentially looping over all density map values and storing
the minimal and maximal value. This has a time complexity of $O(m^2)$, where $m$ is the grid-size
(we look at each cell). The ping-pong approach actually has a $O(m^2 \log m^2)$ time complexity.
However due to the parallel nature of GPU programming the latter solution is faster in practice.

5.3 Rendering Regions

Up until now we have only conceptually talked about regions and not made explicit how we
actually render them. One option is to do this vector-based: given a region to be displayed, we can
trace its bounding polygon from the Delaunay triangulation (this is explained in Section 5.4.1)
and render it using OpenGL. Another approach is to do it pixel-based: for each pixel individually
we choose a color, based on information available from the density map. Intuitively this can be
seen as rendering a slice of the density map.

Definition 10. We call the boundary of the set of triangles partially contained
in the region $R$ (see Definition 8) the bounding polygon of $R$ (Fig. 5.11).

The obvious drawback with the pixel-based approach is that it is not
straightforward to render a single region in isolation. We can only ren-
der all regions at a given density. An advantage is that it gives fine-grained
control over the resulting graphics, because we control the coloring of each pixel individually. Also, applying more smoothing to the density map will automatically result in smoother region shapes in the visualization. On the other hand, the latter might also be considered a drawback, because it gives less control when we directly want to influence the complexity of the shape (methods to do this do exist however, e.g., wavelet descriptors [12]). Using a vector-based approach, we have more direct control over region shape complexity, because we can make the shape less complex by intelligently removing vertices [5], or apply smoothing by using splines instead of straight line segments.

**Hybrid approach.** We want to be able to render each region individually. This is useful for interaction purposes (e.g., being able to select individual regions and show statistics specific to this region), but also if we want to use information for the color of the region that is not directly available in the density map (e.g., the weighted number of points in a region). This means we cannot use the standard pixel-based approach. The purely vector-based approach is also not preferred, because it means we have to do extra work to get smoother shapes. We use a hybrid approach, which uses the bounding polygon of the region as a *canvas* to render the regions on (Fig. 5.12). This means we still have the ability to render each region individually using a pixel-based approach. We address the subtleties of this approach in the remaining part of this section.

![Figure 5.12: Rendering of a region represented by node $n$ in the contour tree $T$: first, get the set of contained triangles, second, trace the boundary (the bounding polygon), and finally, render the region on this bounding polygon (pixel-based).](image)

### 5.3.1 Region ordering

Essentially our visualization of regions is unbalanced. With unbalanced we mean that regions from different levels of density are visualized. Therefore, the shape of a region only communicates the geometric properties of that region, and not necessarily the density. On the map we might show two regions as having equal area, while they contain a different number of weighted points. This also happens on standard geographic maps and is commonly solved by indicating the size of the region using symbols. Another possibility is to communicate ordering by coloring the regions.

**Color.** Instead of a single color to represent regions on a map, e.g., green for vegetation and gray for industrial areas, we can use multiple color shades to indicate properties of regions. Regions can be colored based on a number of attributes, for example, the weighted point count, and the minimum, maximum, or mean density. Generally, we use a darker shade for higher values of the attributes. This follows our intuition that regions with higher attribute values are more important, and as a result more noticeable on the map, because of the higher contrast with the background (assuming this is white). By default we show the weighted point count attribute, however, we have to be careful, because it might work well for actual population data, but can also give a wrong impression, because important regions can dominate the map if they become big (area) and also have a dark shade (high number of points).
**Boundary artifacts.** Using this method to render regions can cause boundary artifacts at positions on the map where regions are about to merge, but are still separate. This happens because parts of the other region will be inside the bounding polygon. We could reduce these artifacts by not considering inserted points (Section 4.4.1) as a seed point for possible regions. This also means that the transition of the two regions splitting will be also less gradual. Another solution would be to apply the same flood-fill approach to the triangle where this happens. However we have not implemented this as in practice this is only a minor problem for our experimental setup.

### 5.3.2 Boundaries

Rendering the fill of a region $R$ is straightforward: color each pixel if it has a density greater than, or equal to, the density of the lowest density point in $R$. Rendering the boundary of $R$ is more difficult. We want the profile of the boundary to align precisely with the edge of the region, and therefore, render it based on the density map as well. Furthermore, for aesthetic reasons, we want the boundary to have an equal width at any position. This requires some extra work.

**Equal width.** Let $F = (f, \mathbb{R}^2)$ be the scalar field that represents the density map. The boundary should be rendered with its profile exactly on the edge of the region, i.e., the isoline at density $d$, where $d$ is the density of the region. We start by looking at a point $(x_0, y_0)$ in the vicinity of the isoline (Fig. 5.13) and define the first order approximation $F$ of $f$ in the vicinity of $(x_0, y_0)$ to be

$$F(x_0 + u, y_0 + v) = f(x_0, y_0) + \nabla f \cdot (u, v), \quad (5.8)$$

where $u, v \in \mathbb{R}$ and $\nabla f$ denotes the gradient of $f$.

Our goal is to find the point $q = (x, y)$ closest to point $(x_0, y_0)$ that has density $d$. If this point has a distance smaller than $w^2$ (where $w$ is the desired width), we know we have to color it, because it falls within the boundary.

To find $q$, we use the gradient of $f$ at $(x_0, y_0)$. This gradient points in the direction of the greatest rate of change at $(x_0, y_0)$ of the scalar field (by definition). So we should be able to find $q$ by solving $F(x_0 + u, y_0 + v) = d$ and setting $(u, v) = t\nabla f$, where $t \in \mathbb{R}:

$$F(x_0 + u, y_0 + v) = f(x_0, y_0) + \nabla f \cdot t\nabla f = d. \quad (5.9)$$

Solving this we get

$$t = (d - f(x_0, y_0))/ (\nabla f \cdot \nabla f). \quad (5.10)$$

We can write the squared euclidean distance $r^2$ in terms of $u$ and $v$ as

$$r^2 = u^2 + v^2 = t^2 (\nabla f \cdot \nabla f) \quad (5.11)$$

Substituting $t$, we get

$$r^2 = (d - f(x_0, y_0))^2/(\nabla f \cdot \nabla f). \quad (5.12)$$

Therefore, the euclidean distance $r$ to the closest point $q$ with density $d$ is defined as

$$r = |d - f(x_0, y_0)|/|\nabla f|. \quad (5.13)$$
Given a width \( w \) we can now render the boundary by looking at each pixel individually. We compute \( r \) using the center of the pixel and define the function \( I(r, w) \) to denote the opacity of the pixel as

\[
I(r, w) = \begin{cases} 
1 & \text{if } |r| < \frac{w}{2} \\
0 & \text{otherwise.} 
\end{cases}
\] (5.14)

This will result in a boundary of width \( w \) around the profile of the region.

**Anti aliasing.** The above definition of \( I \) results in sharp edges that will look jagged. To resolve this we apply anti aliasing. We redefine the function \( I \) to

\[
I(r, a, b) = \begin{cases} 
1 & \text{if } |r| < a \\
(1 - \frac{|r| - a}{b - a})^2 & \text{if } a \leq |r| < b \\
0 & \text{otherwise,} 
\end{cases}
\] (5.15)

where \( a \) is the width of the core of the boundary (which is solid), and \( b \) is the width of the total boundary (Fig. 5.14). This means we use \( b - a \) portion of the boundary for anti aliasing (by simple linear interpolation of the opacity). By default we use \( a = \frac{1}{8}w \) and \( b = w \), where \( w \) is the chosen width of the boundary.

**Boundary types.** There are a number of possible boundary types. Besides the default equal width boundary, we can also choose two types of fuzzy boundaries, which can be used to experiment with visualizing the uncertainty of the placement of the boundary. For the first type we use the thickness of the boundary to indicate the certainty and for the second type the opacity. Both are simple to implement and require just a slight variation on the method described above.

For both types we use the surface normal \( n = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, 1) \) at point \( p \) to get a measure of the certainty. Intuitively, a steep slope in the density map indicates a higher certainty that the boundary should be placed there, while a more gradual slope indicates that the boundary is equally likely to be anywhere in the vicinity. As a measure, we use the angle \( \theta \) (in radians) between the vector \( u \) pointing upwards \((u = (0, 0, 1))\) and the surface normal \( n \). For a flat surface \( \theta \) will be 0° and for an for an infinitely steep slope it will be 90°. We use the dot product to compute \( \theta \):

\[
\theta = \cos^{-1} \left( \frac{n \cdot u}{|n| \cdot |u|} \right), \quad \text{and since } n \text{ and } u \text{ are both normalized, } \\
= \cos^{-1}(n \cdot u).
\] (5.16)

We define the certainty \( c \) as the normalized angle \( \theta \), formally \( c = |\theta|/\pi \). For the thickness type we can simply make the width dependent on \( c \), and for the opacity type we can define the new opacity \( I' \) by multiplying the old opacity, \( I'(r, a, b) = I(r, a, b) \cdot c \).

Note that we only visualize the uncertainty of the position of the boundary itself, and not the uncertainty of the underlying data (e.g., due to measurements errors). Research exists on this as well [27].
5.4 Contour tree

Section 4.4.1 describes how we improve the input for the contour tree algorithm, such that the resulting contour tree is more consistent with our actual density map. Recall that the original input for the contour tree is the scalar field \( F^* = (f^*, M) \). Here \( M \) is a mesh, required to be a simplicial complex, defined over the input points \( P \) and \( f^* \) is assumed to be a piecewise-linear interpolation function over all simplices of \( M \). The scalar values at the vertices of \( M \) itself are directly taken from the density map \( F = (f, R^2) \). For \( M \) we use the Delaunay triangulation \( T \).

Locating min and max points. Locating the min and max points is done by looking at each edge \( e \in M \). For each edge \( e \) we look at the line-segments \( L \) that are formed by the intersection of \( e \) with the quads formed by the centers of the grid points from the density map \( F \) (Fig. 5.15). Recall that values for points not at grid centers in \( F \) are computed using bilinear interpolation. Our goal is to find the points on \( e \) that have the minimal and maximal interpolated value within such a quad \( Q \). We can derive function \( \text{minpoint}(l) \) and \( \text{maxpoint}(l) \) that do this for a line-segment \( l \in L \) based on properties of bilinear interpolation (for the derivation see Section D.1).

Using these functions we locate the actual min- and max-points by iterating over all quads \( Q \) intersected by \( e \) and keeping track of the minimal and maximal points. We do this for all edges \( e \in M \), and store these points in a set \( P^* \). For the union of the new set \( P^* \) and the original input \( P \) we recompute the Delaunay triangulation and use this as an input for the contour tree algorithm instead.

Notes on the time complexity. By Euler’s formula, we can express the number of directed edges \( e \) in a triangulation as \( e = 3n - 3 - k \), where \( k \) is the size of the outer face. This means \( e = O(n) \). We give a naive upper bound on the length of an edge by reasoning that an edge is always a straight line. Therefore, the maximum length edge within our grid is the diagonal of the grid. By Pythagoras’ Theorem we know this to be \( \sqrt{m^2 + m^2} = m\sqrt{2} \), where \( m \) is the grid resolution. For each edge \( e \) we see each grid cell intersecting \( e \) once. This results in a total time complexity of \( O(nm) \), however in practice this will be much better (shorter edges).

5.4.1 Computing the bounding polygon

As discussed in previous sections, we use the bounding polygon of a region \( R \) as a canvas to render \( R \) on. We have all individual triangles of this polygon, as we can extract these from the triangulation using the contour tree. This means that we could already do the rendering by simply drawing on each triangle individually. However, as discussed in Section 5.4.2, we want to be able to do operations on the bounding polygon such as expanding it. To do this directly on the set of triangles, without making them overlap, would be overly complex. It pays off to first trace the boundary of this set of triangles (i.e., the bounding polygon) and use this instead. In this section we outline the algorithmic details on how this is done.

Definition 11. We say an edge \( e \) in the Delaunay triangulation \( T \) is inside a region \( R \) if both triangles in \( T \) that share \( e \) are contained in \( R \). If both are outside \( R \) then \( e \) is said to be fully outside of \( R \).

First we have to find a candidate vertex \( v \), that is on the exterior boundary of the region \( R \). We take the vertex in \( R \) that has the lowest \( x \)-coordinate.

Definition 12. We say an edge \( e \) is on the boundary of a region \( R \) if and only if exactly one of the triangles in \( T \) that share \( e \) is contained in \( R \).
The bounding polygon is computed by tracing the boundary of $R$. Starting at $v$, we look for the next edge by clockwise iterating over its neighbors. If we find the edge $e$ that has its counter-clockwise triangle inside of $R$ (from the perspective of $v$) and the clockwise triangle outside of $R$, we know $e$ to be on the boundary of $R$. We proceed by adding $e$ to the bounding polygon. This procedure is repeated until we see the same directed edge twice (Fig. 5.17), this means the polygon is closed.

Note that it is important that we walk along the inside of the boundary, because otherwise it is possible that we completely miss parts of the polygon (as illustrated in Fig. 5.16). The opposite case (imagine the inverse of Fig. 5.16) does not cause any problems, because if $p$ is contained in $R$ then all triangles containing $p$ will be part of the bounding polygon. This means that for each edge $e$ we see on the boundary, there is always a triangle touching $e$ on the interior of the bounding polygon.

Notes on the time complexity. The time complexity of this algorithm is $O(n)$, where $n$ is the number of vertices of the triangulation. This follows from the fact that during the boundary trace we see each directed edge in the triangulation at most once. According to Euler’s formula we can express the number of directed edges $e$ in a Delaunay triangulation as $e = 3n - 3 - k$, where $k$ is the size of the outer face. So $e = O(n)$ and thus the algorithm is $O(n)$.

Lemma 1. For any possible set of non-nested regions $S$, represented by a set $S'$ of nodes selected from the contour tree $T$, where for each node $n \in S'$ there is no ancestor of $n$ also in $S'$, it holds that each triangle is overlapped by at most three bounding polygons of regions in $S$.

Proof. This follows from the fact that all regions in $S$ have a disjunct set of points (by selection). A triangle is overlapped by a the bounding polygon of a region $R$ if it is either fully or partly contained in $R$. This means that $R$ covers at least one of the vertices of the triangle. If we assume that the number of overlapping bounding polygons is more than three for a triangle, it means that there are at least two regions that cover the same vertex. This is impossible due to the disjunctness of the sets of points. □

By Lemma 1, we can improve on the $O(n)$ bound by showing that the time complexity remains $O(n)$ for the entire region selection, instead of just a single region. We reasoned that the time complexity for a single region is $O(n)$ because we visit each directed edge only once. To show the time complexity for the entire region selection we have to look at how often this edge is revisited when computing the bounding polygon for all regions in a selection $S$. There are two possibilities:

1. We look at a triangle not seen before that touches $e$ (at most one because an edge is shared by exactly two triangles), and;
2. We look at a triangle seen before which contains $e$ (at most three by Lemma 1).
This means for the entire region selection we see a directed edge at most 6 times, which means the time complexity remains $O(n)$.

**Holes.** Regions might contain holes. Consider for example a set of points all placed at a distance $r$ from a center point $p$ (i.e., on the boundary of a circle with center $p$ and radius $r$). Plotting these points would create a single circular region with a hole in its center. Ideally the bounding polygon should contain these holes as well. This can be done by not only tracing the exterior boundary, but all interior boundaries as well (and then taking the complement of all interior boundaries in the exterior boundary). We however have not done this, because in general we assume regions on a map not to contain any holes.

### 5.4.2 Expanding the bounding polygon

Because it is possible that parts of a (thick) region boundary are outside of the region we have to make sure the bounding polygon is large enough to contain parts of the boundary that are outside of the region as well. We do this by expanding the bounding polygon to make sure it fits. If the boundary is set to be rendered outside of the region (Fig. 5.18(b)) we expand by $w$, where $w$ is the width of the region. If it is rendered with its profile on the edge of the region (Fig. 5.18(a)) we expand by $\frac{w}{2}$.

The expansion is done by iterating over the vertices of the bounding polygon (in order). For each vertex $v$ we look at the edges leaving $v$. We distinguish three cases: (1) the edges are parallel (up to a certain threshold $\epsilon$ to allow for numerical errors), (2) the edges face inward with respect to $v$, and (3) the edges face outward with respect to $v$.

Let $e_r$ and $e_l$ be the edges on the right- and left-hand side of $c$ and on the border of $C$ respectively, and let $w$ be the expansion width.

1. **$e_r$ and $e_l$ are near parallel** $\left(180^\circ - \epsilon < \alpha < 180^\circ + \epsilon\right)$.
   
   We replace $c$ by $c_t$, where $c_t$ is the $c$ translated a distance $w$ over the vector perpendicular to the two edges and facing outside of $C$.

2. **$e_r$ and $e_l$ are facing inward with respect to $c$** $\left(\alpha < 180^\circ - \epsilon\right)$.
   
   Let $v_r$ and $v_l$ be the edges perpendicular to $e_r$ and $e_l$ respectively, both facing outside of $C$. We compute new line segments $l_r$ by translating $e_r$ a distance $w$ over $v_r$ and analogue for $l_l$ and $e_l$. We replace $c$ by the intersection of the two translated lines.

   A special case occurs when the line-segments $l_r$ and $l_l$ do not intersect. This means that the boundary will fill the entire gap. We solve this by simply discarding $c$.

3. **$e_r$ and $e_l$ are facing outward with respect to $c$** $\left(\alpha > 180^\circ + \epsilon\right)$.
   
   Similar to the inward case let $l_r$ and $l_l$ be the line-segments translated over a distance $w$ in the direction of $e_r$ and $e_l$ respectively. Let $l$ be the line-segment perpendicular to the normal at $c$ translated $w$ outside of $C$.

   We replace $c$ by the two intersection points $c_{11}$ and $c_{12}$ of the lines going through $l_l$ and $l_r$ with the line-segment $l$.

**Notes on the time complexity.** We iterate over the vertices of the bounding polygon once, doing only $O(1)$ amount of work for each vertex that we see. Therefore, the time complexity of the entire expansion is linear in the size of the bounding polygon. This size is $O(n)$, where $n$ is the number of vertices in the input. This follows from the time complexity of computing the bounding polygon. This bound can be improved to $O(n)$ time complexity for the entire region selection with
an argument similar to the discussion on the time complexity of computing the bounding polygon. For this discussion we refer to Section 5.4.1.

5.5 Map layers

As is very common in Geographic Information Systems, we generate different layers for each different set of map features (e.g., context, regions and region symbols). This way we can compose and experiment with different maps by showing, hiding and reordering these layers. We briefly enumerate the layers currently available. Some of these are discussed in more detail later sections.

1. **Density map 1 and 2**: Show a plot of the first or second density map by mapping the density value to a color (using a colormap).
2. **Base map**: Contains the base map (i.e., administrative boundaries) of certain regions (the world, Europe, the Netherlands). These boundaries for each region are stored in vector format using ESRI ShapeFiles (which is a popular format for distribution of this type of data). Reading ShapeFiles is done using a 3rd-party library called ShapeLib.
3. **Regions fill**: The interior fill of the regions.
4. **Region boundaries**: We have chosen to put the region boundaries in a separate layer, which gives the possibility to render them separately.
5. **Points**: Show the original point set (i.e., the scatter plot).
6. **Shading 1 and 2**: Show hill shading either for the first or second density map.
7. **Per region shading 1 and 2**: Same as the shading layers except shading outside of regions is clipped.
8. **Info**: Show statistics such as number of points, total area covered for the current region selection.
9. **Symbols 1 and 2**: Symbols of the regions for density map/contour tree 1 and 2.

**Base map.** Showing the base map, being the administrative boundaries, can cause problems such as regions overlapping with the boundaries. In cartographic generalization it is also common the reduce, or increase, the amount of detail for these boundaries (i.e., their complexity) depending on the scale of the map [37]. However, because this is not the main focus of our work we consider the base map fixed (no generalization) and we consider overlap of regions with the boundaries to be allowed.

5.5.1 Colormaps

As already described in Section 5.3.1, we use color to indicate the importance of a region. There are a number of region attributes that we can consider to denote its importance; the weighted point count, region density, highest density in region and mean density. We use a colormap, or a color table, to map values for these properties to a color. A colormap is defined as a set of seed colors, where each color has a related position in the colormap (between 0 and 1). At least the colors at the endpoints (0 and 1) have to be defined. From this we generate a colormap by linearly interpolating the color between each position (linear gradients).

A value for an attribute is mapped to a color using: (1) *absolute scaling*, between zero and the maximum value for this attribute, or (2) *relative scaling*, between the minimum and maximum value of this attributes among the regions currently visible on the map. In practice, using the absolute value often means that there will be little variation in color, because the actual range for most of these attributes in the regions currently visible on the map will be small compared to the full range of these attributes. As a solution we can use the relative scaling method instead.
However, this has the unfortunate side-effect that we sometimes see abrupt changes in region colors if we change the set of regions currently visible (e.g., by changing the selection strategy, or parameters of the selection strategies). We will commonly use lightness of the color to indicate the importance of a region (darker is more important).

We also use colormaps to directly visualize the density maps. This can be seen in Figure 4.2 using a gray-scale colormap (white on position 0 and black on position 1).

5.5.2 Symbolization

On standard topographic maps, the shape of a region primarily communicates the geographic properties of that region (e.g., area, boundary, position). Generally speaking, more important regions are more dominant on the map, because of their larger area. However, if two regions are approximately the same size on the map, it might be the case that one of these regions actually contains more points (has a higher average density). To indicate this, simple symbols are used that classify regions into groups (e.g., large, medium, small, etc.).

If we make the symbols more prominent on the map (by hiding the regions and increasing the size of the symbols), we can also make proportional symbol maps (Fig. 5.19(e)).

**Placement and size of symbols.** The placement and sizing of these symbols is a problem which has been covered frequently in cartographic literature. We want to experiment with adding symbols to our maps, however, since this is not the main focus of our work, we use a fairly straightforward approach for the placement and sizing, and show only symbols for regions that are selected to be displayed on the map as well (i.e., there are no symbols indicating outliers).

The size of a symbol is based on the weighted point count for a region, using a polynomial scale. The value for this exponent and the base size of the symbols are configurable from within the user interface. Figure 5.19 illustrates the effect of changing this base size. We give two options for the placement of this symbol, the first is the location of the densest point within the region (under the assumption that this point is central to $R$) and the second is the center of the bounding box of the points contained in $R$. Both of these are easy to compute given data available from the contour tree (Section 4.4.2).
5.5.3 Hill shading

Hill shading is a technique to visualize the topography of a landscape by illuminating its slope. Using a light source, typically from the North East, areas facing away from the light source are shaded darker, while areas directly facing the light source are illuminated (Fig. 5.20). As shown by Wise et al. in their ThemeScape map viewer, this technique can effectively be used to subtly hint density in a contour map [52], and also more explicitly, by van Wijk et al. [47]. We implement this technique using per-pixel Phong lighting [28].

Examples. Experimenting with different values, we can create a number of different visualization, varying from very subtle to very explicit density hinting (see Fig. 5.20).

Figure 5.20: Illustration of hill shading on Brabant using a 2.5 km kernel bandwidth: (a) using a low ambient intensity, no specular component and high diffuse intensity; and (b) using a high ambient intensity, high specular intensity and high diffuse intensity

5.5.4 Exporting

Comparing the effect of different methods and varying parameter values can be tedious if we have to do this by hand. It is useful to be able to quickly generate overviews of results using various methods and varying parameter values. We introduce a basic scripting language, which allows us to programatically change settings, and perform actions such as taking snapshots and re-computation of the density map, contour tree and other required data structures (see Figure 5.21 for an example script). A detailed list and explanation of all commands can be found in Appendix B (Section B.1).

Variables. Variables can be used to store values, or as iterator in a loop. In general the value of a named variable name can be accessed by using <name>.

Output. The output of the batch script (i.e., a number of sequences of snapshots) can be exported in two different formats: in a grid or as an animation (video). Within the grid, each column represents a sequence of snapshots, and each row represents a new snapshot within this sequence. For the animation each snapshot is considered a single frame of the animation. The sequences themselves are animated sequentially.

Settings. For an overview of all available settings see Appendix B (Section B.2).
Chapter 6

Evaluation

In Chapter 3 we have discussed - in our opinion - relevant properties of the type of map that we pursue, and provided a model (Chapter 4), and prototype (Chapter 5), to automatically generate different varieties of these maps. In this chapter we evaluate the model by discussing results of a user study that we have performed on 17 test subjects. Recall part of our problem statement as given in Chapter 1:

*What are relevant properties that characterize such regions, how can we generate regions according to different valuations for these properties, and what are the preferences of people for values?*

This user study is an attempt to make this user preference more explicit. More specifically

*Can we say anything tangible about user preference on properties of the type of map we generate?*

To keep this user study compact, we focus on the three main aspects that define the aesthetics of our maps: the selection strategy used (i.e., the distribution of regions), the kernel bandwidth, and the percentage of map covered.

**Setup.** From a high level point of view, our user study is structured as depicted in Figure 6.1. For each strategy, we present the user with a number of possible representations. Based on a reference image (depicting density of an original set of points), and the task (in the form of a question), the user picks his or her favorite representations. This choice can be mapped back to specific parameter values, and thus, gives us per user, per test case, the preferred parameter values.

![Figure 6.1: Structure of the user study.](image)

For a more detailed discussion on the setup of the evaluation as well as the user interface of the actual user study that was performed see Appendix A.

**Test data.** As a source of data for the different test cases we use two main datasets:

- **LandScan global population database 2008 (LandScan2008) [1]:** Contains actual populated density data for the entire world, measured in a grid of 30 by 30 arc seconds (approximately one square kilometer).
- **Paris500k [42]:**
  Contains the locations of 500,000 geotagged photos in the city center of Paris.

From these two datasets we extract a total of twelve different regions (Table 6.1). All these regions represent a single test case, for which we test both the constant density and the map coverage strategies. Note that we use the same regions (Brabant, The Netherlands and Western-Europe) twice. For the first three test cases, we include the base map (marked with an *), and ask a question specific to the distribution of the points. For the second three, we omit the base map, and ask a question related to the patterns and characteristics of the data in general. These questions are given in Table 6.2.

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Source</th>
<th>Size</th>
<th>Context</th>
<th>Physical size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Brabant*</td>
<td>LandScan2008</td>
<td>9k</td>
<td>yes</td>
<td>(133.93 km 130.03 km) × 132.13 km</td>
</tr>
<tr>
<td>1</td>
<td>The Netherlands*</td>
<td>LandScan2008</td>
<td>194k</td>
<td>yes</td>
<td>(344.92 km 323.08 km) × 333.96 km</td>
</tr>
<tr>
<td>2</td>
<td>Western-Europe*</td>
<td>LandScan2008</td>
<td>213k (resampled)</td>
<td>yes</td>
<td>(3899.56 km 2204.59 km) × 3029.52 km</td>
</tr>
<tr>
<td>3</td>
<td>Brabant</td>
<td>LandScan2008</td>
<td>9k</td>
<td>yes</td>
<td>(133.93 km 130.03 km) × 132.13 km</td>
</tr>
<tr>
<td>4</td>
<td>The Netherlands</td>
<td>LandScan2008</td>
<td>194k</td>
<td>yes</td>
<td>(344.92 km 323.08 km) × 333.96 km</td>
</tr>
<tr>
<td>5</td>
<td>Western-Europe</td>
<td>LandScan2008</td>
<td>213k (resampled)</td>
<td>yes</td>
<td>(3899.56 km 2204.59 km) × 3029.52 km</td>
</tr>
<tr>
<td>6</td>
<td>US small</td>
<td>LandScan2008</td>
<td>40k</td>
<td>no</td>
<td>(158.65 km 154.80 km) × 156.73 km</td>
</tr>
<tr>
<td>7</td>
<td>US medium</td>
<td>LandScan2008</td>
<td>861k</td>
<td>no</td>
<td>(769.78 km 689.29 km) × 729.59 km</td>
</tr>
<tr>
<td>8</td>
<td>US large</td>
<td>LandScan2008</td>
<td>737k (resampled)</td>
<td>no</td>
<td>(1394.18 km 1160.90 km) × 1278.26 km</td>
</tr>
<tr>
<td>9</td>
<td>Paris small</td>
<td>Paris500k</td>
<td>28k</td>
<td>no</td>
<td>(966 m 966 m) × 966 m</td>
</tr>
<tr>
<td>10</td>
<td>Paris medium</td>
<td>Paris500k</td>
<td>113k</td>
<td>no</td>
<td>(3099 m 3097 m) × 3098 m</td>
</tr>
<tr>
<td>11</td>
<td>Paris large</td>
<td>Paris500k</td>
<td>187k (500k before removing duplicates)</td>
<td>no</td>
<td>(5985 m 5978 m) × 5981 m</td>
</tr>
</tbody>
</table>

*Table 6.1: Details of the twelve test cases used during the user study.*

**Parameter range.** As already mentioned, besides the strategies, the two main parameters we are testing are the *kernel bandwidth* and the *percentage map coverage*. For the map coverage strategy the *spread factor* is included in the user study as well.

Note that we have defined the kernel bandwidth in terms of the physical size of the map on screen, and not in meters. This means that we can directly compare the bandwidth for the different cases.

**Generating the reference images.** The reference image should allow the user to correctly judge density of the original set of points. To do this, we cannot directly plot the points, because the LandScan2008 dataset is sampled on a grid, and for the Paris500k dataset we cannot guarantee that there is no overplotting if we directly plot the points. This means that we first have to generate representations that enable the user to make correct density judgments. We use density plots (Fig. 6.2), generated by smoothing the point data (using KDE, see Section 4.3.1). However,
CHAPTER 6. EVALUATION

<table>
<thead>
<tr>
<th>Test cases</th>
<th>Task (in the form of a question)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 2</td>
<td>On the left you see the population density of Brabant, on the right a schematic representation. In your opinion, which schematic representation best depicts the population distribution.</td>
</tr>
<tr>
<td>3 – 11</td>
<td>On the left you see the density of an original set of points, on the right a schematic representation. In your opinion, which schematic representation best depicts the patterns and characteristics of the original set of points.</td>
</tr>
</tbody>
</table>

Table 6.2: Questions asked for each of the twelve test cases (translated from Dutch).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel bandwidth</td>
<td>0.01</td>
<td>0.06</td>
<td>0.005</td>
</tr>
<tr>
<td>percentage map coverage</td>
<td>0.5%</td>
<td>12.5%</td>
<td>1%</td>
</tr>
<tr>
<td>spread</td>
<td>0.2</td>
<td>0.9</td>
<td>0.35</td>
</tr>
</tbody>
</table>

Table 6.3: Parameters and their range (as used for the user study).

The obvious problem is that by doing this we already possibly bias the user, because we already choose a value for one of the parameters that we are actually investigating: the kernel bandwidth. We ignore this problem for now, and discuss it after the discussion on the results of the user study (Section 6.3).

Figure 6.2: Illustration of the effect of applying different scaling to the reference image: (a) linear, (b) polynomial \(x^{0.8}\) and (c) logarithmic.

A second problem is the scaling that we apply to these density plots. As illustrated in Figure 6.2 and Figure 6.3, using a linear scale versus a logarithmic scale has a significant impact on the density plot. The linear scale only reveals the absolute hot spots (Fig. 6.3(a)), while the logarithmic scale makes these less prominent, and more clearly shows lower density regions (Fig. 6.3(c)). The differences become more significant as the scale of the map increases (Fig. 6.3). This is because typically for population data, on a larger scale, the differences between densities will be larger.

For the user study we use the polynomial version, using 0.8 as exponent. We also reduce the dominance of the hot spots on the map slightly by clamping the scaling at 90%.

Relative density strategy. After performing a small pre-study, we found that the relative density strategy caused some unpredictable results, and in general, users found it harder to find a good representation than for the other two strategies. Therefore, we made the decision to omit it from the actual user study.

Structure. We evaluate the results by looking at both parameters (kernel bandwidth and coverage) individually. An important aspect is the consistency in chosen values. Ideally, when all users pick the same map, we know this to be the optimal map and we're done. However, in reality the test is really subjective, and will probably result in high variation. One reasonable assumption that we can make, is that users individually will be more consistent than the group as a whole. Also, for each individual test, consistency within this test is more likely to be higher than compared to
all groups as a whole. This is because of the specific characteristics of the data. To confirm this, we also specifically discuss the results per individual user and per individual test.

Another interesting question is to see how the two strategies perform in relation to each other. Therefore, we will also group results based on the strategy during the discussion. For the first six tests, the context is known, and we make the assumption that the test subjects have a fairly good knowledge of the topography of these regions. For the last six it is not. Based on this assumption, we also split the data into the following two groups: context known and context unknown.

**Plotting the results.** First, to get an impression of the results of the user study, we plot the bandwidth and coverage using scatter plots. Figure 6.4(a) illustrates the different variations of maps as a result of increasing the bandwidth (along the y-axis) and increasing percentage of map covered (along the x-axis). The actual values for the user study are plotted for the constant density strategy (Fig. 6.4(b)), and map coverage strategy (Fig. 6.4(c)). The range depicted in these scatter plots (0.01 to 0.06 for bandwidth, and 0.05% to 12.55% for coverage), which is exactly the range that we have used for the evaluation (i.e., there were no choices outside of this range).

These images suggest that, on average, users prefer a lower bandwidth and do not have a strong preference for the coverage. The Pearson correlation coefficient between the two parameters is 0.097, which indicates that they not strongly correlated. In the following sections we discuss these observations in more detail. To get some idea about the actual visual difference between images at the boundaries of the grid, examples are shown in Appendix C (Fig. C.13).
CHAPTER 6. EVALUATION

6.1 Coverage

We start with a brief summary of the descriptive statistics for the coverage parameter. Figure 6.5(a) plots the frequencies of preferred values. In Figure 6.5(b) we list the mean and standard deviation of the coverage, where we group the test cases based on the regions they depict. To get an idea of the consistency for individual users, and on individual tests, we give the average standard deviations per method as well (Figure 6.5(c)). As an attempt to make these statistics more concrete, three maps are shown (Figure 6.6). These maps are generated using a coverage based on the mean and average standard deviation for individual users.

![Figure 6.5: Descriptive statistics of the coverage parameter for different subgroups: (a) frequencies of preferred parameter values for all results, (b) the mean and standard deviation for the two strategies per test case, and (c) the mean and the mean standard deviation per user, per test and globally (see Table C.1 and Table C.2).](image)

![Figure 6.6: Illustration of the visual difference in varying the map coverage (using the constant density strategy) by the average standard deviation of individual users of 2.866% around the mean map coverage of 6.152% using the mean kernel bandwidth of 0.017: (a) 3.3% coverage, (b) 6.2% coverage, and (c) 9.0% coverage.](image)

The variation in preferred coverage is large (Fig. 6.5(a)). This could be the result of a large variation in preference per user, or in preference per test case. However, if we plot the preferred coverage per user (Fig. 6.7) and per test case (Fig. 6.8), we see that this is probably not the explanation. The averages do differ per user and per test (Fig. 6.5(c)), but in all cases the variations are large, and many of the confidence intervals overlap.

From Figure 6.5(b) and Figure 6.5(c) we can see that the means for the no context versus context seem to be different. The means for the two strategies are about equal, and the standard deviations for both the scale and strategies do not differ much as well. We perform t-tests to check for equality of means between the two groups, context and no context (Table C.6), and the two strategies (Table C.4). For the context we see that the means are different (significance = 0), but for the strategy there is enough evidence to assume them equal (significance = 0.315).

Conclusion. To conclude, we found that users on average preferred a coverage of about 6%, but found that there is also a strong variation between preferences of users and for different tests.
6.1. COVERAGE

Figure 6.7: Box plots showing distribution of the preferred coverage per user (x-axis) and for each strategy (darker shaded is constant density and lighter shaded map coverage).

Figure 6.8: Box plots showing distribution of the preferred coverage per test (x-axis) and for each strategy (darker shaded is constant density and lighter shaded map coverage).
6.2 Bandwidth

Again, we plot the frequencies of the values (Figure 6.9(a)), and list some descriptive statistics for
the bandwidth, based on different groups within the data (Figure 6.9(b) and Figure 6.9(c)). To
make these statics more concrete, we show maps generated using the mean and average standard
deviation for individual users (Figure 6.10).

![Figure 6.9](image1)

![Figure 6.10](image2)

For the bandwidth parameter, the results seem somewhat more promising. Figure 6.4 suggests that
users prefer lower bandwidths in general. Looking at the frequencies (Fig. 6.9(a)), the variation
for the bandwidth seems to be a lot smaller compared to the coverage.

Figure 6.9(b) suggests that there is no difference in test cases between mean values and stan-
dard deviations. However, among the strategies, the consistency for individual users, as well as
on individual tests, seems a lot higher (Fig. 6.9(c)). Visually, by looking at the plots per test
(Fig. 6.11), the confidence intervals for the first strategy are also smaller. Looking at the t-tests
for equality of means (Table C.4 and Table C.6), we see that the probability that the means for
the two strategies are equal is really low (significance = 0). On the other hand, between the test
cases, this probability is statistically significant (significance = 0.054).

Conclusion. To conclude, bandwidth seems to be independent of the percentage map coverage
used. Lower values are preferred, and for the constant density strategy, users are fairly consistent
in choosing the parameter value.
6.3 Strategy preference

Figure 6.12 shows the preference of users for the two tested strategies. For each test case, we compare the rating for the first strategy to the second strategy. We count the number of times the first strategy is preferred over the second (dark line) and vice versa (light line), as well as when they score equally high (dashed line). Taking the total sum, we get 130 cases where the constant density strategy, 41 for the map coverage strategy, and 33 cases where both methods score equal. We conclude that users clearly prefer the constant density strategy over the map coverage strategy.

Generally speaking, we see that users are also more consistent in choosing their favorite representation for the constant density strategy. This possibly has to do with the setup of the user study. The constant density strategy only varies two parameters (bandwidth and coverage), while the map coverage strategy also has the spread parameter interleaved into the $x$-axis. This makes it less predictable, and therefore, harder for the test subjects to find their favorite representation.

Originally we made the assumption that for the first three test cases people might prefer the map coverage strategy. This is because we assume the test subjects to be familiar with the topography of these regions, and the task was to pick the map which they thought best depicted the population distribution. From the off-line discussion after the user-test, the test subjects generally gave three main reasons why they did not prefer the images generated by the map coverage strategy (in comparison to the constant density strategy):

1. Sometimes the representations are too noisy;
2. In the generated representation, regions are visualized at locations where there does not seem to be any density (according to the reference image);
3. Although, smaller regions are depicted better, the consequence of this is often that the more important hot spot regions become less noticeable.

**Similarity to reference image.** From these statements we observe that users might try to pick representations that are very similar to the reference image, which is in favor of the constant density strategy. To see to what extent this is true, we perform a small experiment. In this experiment we compare similarities to the reference image between the group of possible options (the grid) and the set of chosen images by the users.

For each possible option, i.e., each image possibly seen by a subject, we compute the similarity with the reference image. As a measure, we use the structural similarity (SSIM) index [49]. Within a test we normalize the indexes of all images, by scaling them between the maximal and minimal structural similarity index of the images within that test. This is done because for some tests, the images might be inherently less similar to the reference image (e.g., because of the base map that is added or the region boundaries that will be more dominant for datasets with a lot of smaller regions).

We create buckets for each percentage of normalized similarity, and count the number of choices per bucket, as well as the total number of possible options (from the grid) within this bucket. To be able to directly compare the two statistics we again normalize by dividing both by the total number of images within each grid. For the set of chosen images this is 204 (because we performed $17 \cdot 12 = 204$ tests, and for the set of possible options this is 8112 (the total number of images possibly shown to a single user during the user study). The final result is plotted in a graph (Fig. 6.13). The Pearson correlation coefficient between the two is 0.772. The main conclusion that we draw, from both the correlation coefficient and the graph, is that there is no evidence that suggests users tried to pick representations similar to the reference image.

**Hot spot regions.** One of the main advantages of the constant density strategy is that it portrays the hot spots - as they are visible in the reference image - very well. However, it does not always show smaller regions in sparse area, which might have a high local importance. The map coverage strategy tries to achieve this. A lot of subjects indicated, that as they tried to get some smaller regions visible on the map, the more important region became less prominent. We identify two reasons that could cause this:

1. All regions have thick dark-gray borders, independent of their size;
2. The colormap that was used has a relatively small resolution, which means we can get a higher contrast between the different regions if we use a colormap with a higher resolution.

We try to improve on this by changing some of the default properties of the visualization (Fig. 6.14). By only dimming small regions, in combination with the colormap we have used and thick dark-gray borders, we get a very chaotic image (Fig. 6.14(a)). Hiding the smaller regions instead, and
using a different color for the borders that is based on the color of the region itself, creates a map that is aesthetically more pleasing (Fig. 6.14(b)). However, some of the regions are still not as prominent as we would like (Leeuwarden, Groningen, Eindhoven, Nijmegen, Maastricht, etc.). Increasing the resolution of the colormap improves this somewhat (Fig. 6.14(c)). We can further improve this by using a different attribute to determine the color of a region. Using the maximum density within a region, instead of the weighted point count, further reveals these hot spots (Fig. 6.14(d)).

These variations show that graphical attributes like the range of the colormap and the coloring of the borders of the regions also strongly influence the result, and the perception of what are more and less important regions. To find clear cut rules for these will not be easy. With our user study we hoped to find strong guidelines for choosing values for coverage and bandwidth, but we found that the variation in preferences is large, and seems to depend on the data shown. If other parameters are varied as well, this might probably lead to even more variation.
In this thesis we have developed, implemented, and evaluated a model to automatically generate point set visualizations heavily inspired on the standard topographic map, and what we call the city-metaphor. The implemented prototype supports an experimental approach, which means that we can easily generate the type of map that we pursue (e.g., Fig. 7.1), and vary the different properties that we consider to be important (see Chapter 3).

The model we have developed does not use standard clustering approaches, but integrates density maps with contour trees instead. As opposed to standard visualizations of these density maps, which only visually cluster the points (e.g., contour maps), we can use information available from the contour tree to make this explicit. This means that we can visualize regions individually, and choose exactly what we want to visualize on the map. This has a number of additional advantages such as; interacting with, or displaying information for, individual regions, but also, only visualizing a certain subset of regions of the density map.

By doing a user study, performed among 17 test subjects, we hoped to find strong guidelines for properties such as the shape and size of individual regions. However, we found that the variation in preference is large, and seems to depend on the user, and the data shown. Besides the preference for low kernel bandwidth values (detailed shapes over simple shapes), and consistency in which users choose a value for this parameter, we cannot make any of such observations on the rest of the parameters that we have tested.
7.1 Future Work

Currently, we use the same contour tree and density map for rendering as well as selection purposes. This is not ideal because if we want to select important regions only, one way to go would be to apply more smoothing to the data, but this would also directly mean that the displayed regions will have smoother shapes with less detail. We could think of solutions that use separate contour trees and density maps for display and rendering, but these are not straightforward, because there is not necessarily any relation between contours for these separate versions.

**Outliers.** As opposed to identifying common patterns in the data, in some cases it might be more interesting to locate the rare instances that do not fit these patterns: outliers. In our model we do not really deal with these outliers other than providing an option to either hide or show all of them. Methods exist that rank outliers by assigning a degree of being an outlier to each point (e.g., the Local Outlier Factor (LOF) [8]). We can use such methods to visualize the interesting outliers alongside the common patterns (our regions).

**More sophisticated interaction.** Considering the higher goal, data exploration. The (contracted) contour tree itself compactly visualizes the topology we get by smoothing the point data. Currently, this contour tree is not used for purposes other than selection of regions and storing ancillary information. Because this contour tree is not directly visible in the current visualization itself, we can decide to visualize the contour tree alongside the original map as well. Linking these two might be interesting, because this opens up possibilities for more sophisticated interaction, such as manual adjustment of the map by changing the current region selection (possibly aided by the automated selection strategies).

**Continuous adaptive zooming.** Continuous zooming, for data exploration, would be possible by combining the model and the results from the evaluation (Chapter 6). However doing this continuously would require the density maps, contour tree and all other structure to be recomputed on-the-fly for each level of detail. Currently this is not possible as - especially for large datasets - computation simply takes too long to be done in real-time.

A proposed solution is to precompute a number of density maps and their corresponding contour trees (a scale space). When zooming, we could build an intermediate representation based on the two closest density maps. As observed by van Kreveld [45], intermediate representations are less important than final representations and only aid in the user not losing focus. Either combining the two density maps, creating a kind of morphing effects between the two maps (for example by linear interpolation between the two values), or varying the density in one of the two (or both) allows us to vary the amount of detail shown. After the user stops zooming (given a certain threshold) the final representation can be computed.
Bibliography


In this chapter, we give a detailed description of our method of evaluation. The basic idea is simple, we present the user with data and a task (in the form of a gray-scale reference image, depicting density, and a question about the data, see Figure A.1) and propose a set of schematic images we have generated using the method as described in Chapter 4. We ask the user to pick the image that is most suitable with respect to the data and task.

To keep the experiment compact, we only treat aspects of the model that follow the observations made in the introduction to Chapter 3 (i.e., no density plots, nested regions, shading, fuzzy borders etc). The user study itself is implemented in the form of a web application (Fig. A.1). In this interface the user sees two versions of the data. On the left of Figure A.1 a reference image of the original data and on the right an arbitrary schematic version of the data. The schematic view is actually a grid of images which the user can navigate using the arrow keys (Fig. A.2).

If a user has picked a favorite map for all of the methods he or she will be asked to rate all three of the images.

Alternatives. A possible alternative method is to present the users with a grid of $2 \times 2$ or $3 \times 3$
Figure A.2: Variation in parameters along x and y axis; (a) fixed density, (b) map coverage, and (c) relative density.

generated maps (randomly selected) and ask them to pick their favorite representation depending on the question asked (task) and the reference image. Repeating this a number of times (each time selecting images not shown before), should eventually give insight into the user preference for certain values for parameters. This method might be more reliable than the method we use, because in our method the user always has to actively make a single decision, even though he or she might not be exactly sure which map to pick, or what are considered to be important features. However, the drawback is that performing the user study in this way takes significantly more time if we want to get the same coverage (i.e., number of visualization seen by the user) as high as with our method.
Appendix B

Batch export

B.1 Commands

In this section we enumerate the commands available within a batch script.

<table>
<thead>
<tr>
<th>Command</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resplat</td>
<td>none</td>
<td>Recomputes all data structures (density map, triangulation, contour tree) and the visualization.</td>
</tr>
<tr>
<td>snapshot</td>
<td>none</td>
<td>Take a snapshot of the current visualization.</td>
</tr>
<tr>
<td>startseq</td>
<td>none</td>
<td>Start a new sequence.</td>
</tr>
<tr>
<td>init</td>
<td>name, value</td>
<td>Define a new variable name and initialize it with value.</td>
</tr>
<tr>
<td>set</td>
<td>setting name, value</td>
<td>Sets the setting named setting name to value (this is any settings defined in the system, for example, an algorithm parameter or currently selected dataset). A list of available settings is given in Table B.2</td>
</tr>
<tr>
<td>loop</td>
<td>iterator name, start, end, step</td>
<td>Starts a loop, assigning start to the iterator, and repeats the loop body until the iterator reaches a value greater than end (using increments of step). The value of the iterator variable is accessible from within the loop.</td>
</tr>
<tr>
<td>endloop</td>
<td>none</td>
<td>Marks the end of the current loop.</td>
</tr>
<tr>
<td>inc</td>
<td>name, increment</td>
<td>Increments the variable name by numerical value increment. If the variable does not contain a numerical value its value will simple be considered zero.</td>
</tr>
<tr>
<td>out</td>
<td>string</td>
<td>Adds an extra line to the information printed on the top-left corner of the map. For example, using a counter variable we can use this to add a unique identification number to each snapshot.</td>
</tr>
<tr>
<td>hidelayer</td>
<td>layer name</td>
<td>Hides the map layer named layer name.</td>
</tr>
<tr>
<td>show layer</td>
<td>layer name</td>
<td>Shows the map layer named layer name.</td>
</tr>
<tr>
<td>include</td>
<td>filename</td>
<td>Includes the batch script identified by filename (the filename is relative to the directory the current batch runs from).</td>
</tr>
<tr>
<td>output</td>
<td>destination</td>
<td>Outputs the current grid to the image destination.</td>
</tr>
<tr>
<td>restoresettings</td>
<td>none</td>
<td>Restores all the settings to their original value (before running the batch).</td>
</tr>
</tbody>
</table>

Table B.1: Overview of batch commands and their description

B.2 Settings

In this section we enumerate most of the settings that can be modified from within a batch script. A number of the settings, that are not specifically related to the visualization itself, have not been listed.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kde_dim0_enabled</td>
<td>bool</td>
<td>Whether or not the first density map is enabled (should be computed).</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>kde_dm0_type</td>
<td>int</td>
<td>Kernel type of the first density map, can be Gaussian (0), Epanechnikov (1),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Triweight (2), Variable Triweight DT (3), Variable Brunsdon (4), and Relative</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Density (5).</td>
</tr>
<tr>
<td>kde_dm0_bandwidth</td>
<td>float</td>
<td>Kernel bandwidth for the first density map.</td>
</tr>
<tr>
<td>kde_dm0_rdbandwidth</td>
<td>float</td>
<td>Kernel bandwidth used for the second density map by the relative density</td>
</tr>
<tr>
<td></td>
<td></td>
<td>strategy (see Section 4.5.3).</td>
</tr>
<tr>
<td>kde_dm0_postop</td>
<td>int</td>
<td>Post operation of the first density map, can be none (0), log (1), sqrt (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or inv (3).</td>
</tr>
<tr>
<td>kde_dm1_enabled</td>
<td>bool</td>
<td>Whether or not the second density map is enabled.</td>
</tr>
<tr>
<td>kde_dm1_bandwidth</td>
<td>float</td>
<td>Kernel bandwidth for the second density map.</td>
</tr>
<tr>
<td>kde_dm1_rdbandwidth</td>
<td>float</td>
<td>Kernel bandwidth used for the second density map by the relative density</td>
</tr>
<tr>
<td></td>
<td></td>
<td>strategy.</td>
</tr>
<tr>
<td>kde_dm1_type</td>
<td>int</td>
<td>Kernel type of the second density map.</td>
</tr>
<tr>
<td>kde_dm1_postop</td>
<td>int</td>
<td>Post operation of the second density map.</td>
</tr>
<tr>
<td>kde_result_combineop</td>
<td>int</td>
<td>Density map combination operator, can be div (0), use 1st (1) or use 2nd (2)</td>
</tr>
<tr>
<td>kde_result_postop</td>
<td>int</td>
<td>Post operator for the combined density map.</td>
</tr>
<tr>
<td>display_region_border_type</td>
<td>int</td>
<td>Type of the region borders, can be solid (0), fuzzy opacity (1) or fuzzy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>width (2) (see Section 5.3.2).</td>
</tr>
<tr>
<td>display_point_size</td>
<td>float</td>
<td>Size of points (see Section 5.5).</td>
</tr>
<tr>
<td>display_background_color</td>
<td>string</td>
<td>Map background color (in RGB hexadecimal format #RRGGBB).</td>
</tr>
<tr>
<td>display_border_color</td>
<td>string</td>
<td>Region border color.</td>
</tr>
<tr>
<td>display_border_fromcolormap</td>
<td>bool</td>
<td>Indicates whether or not the border color should be taken from the density</td>
</tr>
<tr>
<td></td>
<td></td>
<td>map.</td>
</tr>
<tr>
<td>display_border_fromcolormap_scale</td>
<td>bool</td>
<td>Can be used to increase, or decrease, the contrast between the fill and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>border of a region.</td>
</tr>
<tr>
<td>display_border_corefrac</td>
<td>float</td>
<td>Denotes the fraction of the border that should be solid. A bigger fraction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>results in less anti-aliasing.</td>
</tr>
<tr>
<td>display_context_exterior_color</td>
<td>string</td>
<td>Color of the exterior context boundary (see Section 5.5).</td>
</tr>
<tr>
<td>display_context_interior_color</td>
<td>string</td>
<td>Color of the interior context boundaries.</td>
</tr>
<tr>
<td>display_context_exterior_bwidth</td>
<td>float</td>
<td>Line width of the exterior context boundary.</td>
</tr>
<tr>
<td>display_context_interior_width</td>
<td>float</td>
<td>Line width of the interior context boundaries.</td>
</tr>
<tr>
<td>display_region_opacity</td>
<td>float</td>
<td>Opacity of a region (fill and border).</td>
</tr>
<tr>
<td>display_context_opacity</td>
<td>float</td>
<td>Opacity of the map context.</td>
</tr>
<tr>
<td>display_points_opacity</td>
<td>float</td>
<td>Opacity of the points.</td>
</tr>
<tr>
<td>colormap_show_attribute</td>
<td>int</td>
<td>The attribute that is used to map a value to a color (see Section 5.5.1).</td>
</tr>
<tr>
<td>colormap_method</td>
<td>int</td>
<td>Colormap mapping method, can be absolute (0) or scaling (1).</td>
</tr>
<tr>
<td>colormap_scaling</td>
<td>int</td>
<td>Colormap scaling, can be linear (0), power (1), and logarithmic (1).</td>
</tr>
<tr>
<td>colormap_scaling_exponent</td>
<td>float</td>
<td>Exponent used in the colormap scaling (if scaling is set to power).</td>
</tr>
<tr>
<td>debug_show_dm_summary</td>
<td>bool</td>
<td>Show a short summary of the computed density maps (small versions of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>first, second, combined density maps and their gradients.).</td>
</tr>
<tr>
<td>debug_show_approx_area</td>
<td>bool</td>
<td>Show the bounding polygon of currently visible regions.</td>
</tr>
<tr>
<td>debug_show_histogram</td>
<td>bool</td>
<td>Show the density histogram of the first density map.</td>
</tr>
<tr>
<td>debug_show_bb</td>
<td>bool</td>
<td>Show the bounding boxes of currently visible regions.</td>
</tr>
<tr>
<td>debug_show_delaunay</td>
<td>bool</td>
<td>Show the Delaunay triangulation of currently loaded point set.</td>
</tr>
<tr>
<td>debug_show_voronoi</td>
<td>bool</td>
<td>Show the Voronoi diagram of currently loaded point set.</td>
</tr>
<tr>
<td>debug_label_nodes</td>
<td>bool</td>
<td>Label nodes in Delaunay triangulation and Voronoi diagram with their unique</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ids.</td>
</tr>
<tr>
<td>debug_show_outliers</td>
<td>bool</td>
<td>Whether or not the show or hide all outliers (points not contained in any</td>
</tr>
<tr>
<td></td>
<td></td>
<td>region currently visible on the map).</td>
</tr>
<tr>
<td>symbols_positioning</td>
<td>int</td>
<td>Positioning of region symbols (see Section 5.3), can be most densest point</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0) or bounding box center (1).</td>
</tr>
<tr>
<td>symbols_size_method</td>
<td>int</td>
<td>The method that is used to scale the symbol size (as a function of its</td>
</tr>
<tr>
<td></td>
<td></td>
<td>weighted point count).</td>
</tr>
<tr>
<td>symbols_size_scaling</td>
<td>int</td>
<td>Relative symbol size scale factor.</td>
</tr>
<tr>
<td>selection_constant_density</td>
<td>float</td>
<td>Constant density value used for the constant density selection strategy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Section 4.5.1).</td>
</tr>
<tr>
<td>selection_map_coverage</td>
<td>float</td>
<td>Percentage of map to be covered (Section 4.5.2).</td>
</tr>
<tr>
<td>selection_mcspread</td>
<td>float</td>
<td>Map coverage selection strategy spread factor (Section 4.5.2).</td>
</tr>
<tr>
<td>selection_area_approx_method</td>
<td>int</td>
<td>The area approximation method (see Section 4.4.2).</td>
</tr>
<tr>
<td>selection recurse_coverage</td>
<td>float</td>
<td>Percentage of coverage for the recursive regions.</td>
</tr>
<tr>
<td>selection recurse_threshold</td>
<td>float</td>
<td>Recursive threshold for map coverage method (in percentage of total area).</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>selection cp_map_coverage</td>
<td>float</td>
<td>Map coverage control parameter (see Section 4.5).</td>
</tr>
<tr>
<td>selection cp_top_num_regions</td>
<td>float</td>
<td>Top number of regions control parameter.</td>
</tr>
<tr>
<td>selection_min_region_area</td>
<td>float</td>
<td>Minimal region area (hide if below this value).</td>
</tr>
<tr>
<td>selection_min_region_wpcount</td>
<td>float</td>
<td>Minimal region weighted point count (hide if below this value).</td>
</tr>
<tr>
<td>shading_phong_ia</td>
<td>float</td>
<td>Phong lighting: (Section 5.5.3): ambient light intensity constant.</td>
</tr>
<tr>
<td>shading_phong_is</td>
<td>float</td>
<td>Phong lighting: specular intensity of the light source (we only use a single light source).</td>
</tr>
<tr>
<td>shading_phong_id</td>
<td>float</td>
<td>Phong lighting: diffuse intensity of the light source.</td>
</tr>
<tr>
<td>shading_phong_ka</td>
<td>float</td>
<td>Phong lighting: ambient reflection constant.</td>
</tr>
<tr>
<td>shading_phong_ks</td>
<td>float</td>
<td>Phong lighting: specular reflection constant.</td>
</tr>
<tr>
<td>shading_phong_kd</td>
<td>float</td>
<td>Phong lighting: diffuse reflection constant.</td>
</tr>
<tr>
<td>shading_phong_a</td>
<td>float</td>
<td>Phong lighting: the shininess constant.</td>
</tr>
<tr>
<td>shading_phong_opacity</td>
<td>float</td>
<td>Opacity of the shading layer.</td>
</tr>
<tr>
<td>shading_clip_context</td>
<td>bool</td>
<td>Whether or not to clip the shading layer at the boundaries of the map context (if possible).</td>
</tr>
<tr>
<td>shading_density_scale</td>
<td>float</td>
<td>Scaling factor of the scalar values in the density map.</td>
</tr>
</tbody>
</table>
Appendix C

User study data

C.1 Additional descriptive statistics

<table>
<thead>
<tr>
<th>Test case</th>
<th>Mean</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CD</td>
<td>MP</td>
</tr>
<tr>
<td>0</td>
<td>7.65</td>
<td>0.31</td>
</tr>
<tr>
<td>1</td>
<td>7.86</td>
<td>0.34</td>
</tr>
<tr>
<td>2</td>
<td>5.83</td>
<td>0.31</td>
</tr>
<tr>
<td>3</td>
<td>9.59</td>
<td>0.35</td>
</tr>
<tr>
<td>4</td>
<td>7.00</td>
<td>0.38</td>
</tr>
<tr>
<td>5</td>
<td>5.28</td>
<td>0.30</td>
</tr>
<tr>
<td>6</td>
<td>7.77</td>
<td>0.34</td>
</tr>
<tr>
<td>7</td>
<td>3.86</td>
<td>0.36</td>
</tr>
<tr>
<td>8</td>
<td>5.35</td>
<td>0.34</td>
</tr>
<tr>
<td>9</td>
<td>3.74</td>
<td>0.33</td>
</tr>
<tr>
<td>10</td>
<td>5.21</td>
<td>0.28</td>
</tr>
<tr>
<td>11</td>
<td>4.68</td>
<td>0.29</td>
</tr>
<tr>
<td>AVG</td>
<td>6.15</td>
<td>0.30</td>
</tr>
<tr>
<td>GLOBAL</td>
<td>5.97</td>
<td>0.36</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test case</th>
<th>Mean</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CD</td>
<td>MP</td>
</tr>
<tr>
<td>0</td>
<td>0.017</td>
<td>0.007</td>
</tr>
<tr>
<td>1</td>
<td>0.024</td>
<td>0.010</td>
</tr>
<tr>
<td>2</td>
<td>0.019</td>
<td>0.010</td>
</tr>
<tr>
<td>3</td>
<td>0.017</td>
<td>0.007</td>
</tr>
<tr>
<td>4</td>
<td>0.021</td>
<td>0.011</td>
</tr>
<tr>
<td>5</td>
<td>0.020</td>
<td>0.011</td>
</tr>
<tr>
<td>6</td>
<td>0.015</td>
<td>0.007</td>
</tr>
<tr>
<td>7</td>
<td>0.017</td>
<td>0.010</td>
</tr>
<tr>
<td>8</td>
<td>0.017</td>
<td>0.008</td>
</tr>
<tr>
<td>9</td>
<td>0.020</td>
<td>0.010</td>
</tr>
<tr>
<td>10</td>
<td>0.022</td>
<td>0.012</td>
</tr>
<tr>
<td>11</td>
<td>0.016</td>
<td>0.007</td>
</tr>
<tr>
<td>AVG</td>
<td>0.019</td>
<td>0.009</td>
</tr>
<tr>
<td>GLOBAL</td>
<td>0.023</td>
<td>0.013</td>
</tr>
</tbody>
</table>

Table C.1: Mean and standard deviation for each individual test per strategy: (a) for the map coverage parameter and (b) for the kernel bandwidth parameter.
### Table C.2: Mean and standard deviation for each individual user per strategy: (a) for the map coverage parameter and (b) for the kernel bandwidth parameter.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Mean CD</th>
<th>Mean MP</th>
<th>Standard deviation CD</th>
<th>Standard deviation MP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.03</td>
<td>5.15</td>
<td>2.10</td>
<td>2.95</td>
</tr>
<tr>
<td>2</td>
<td>9.87</td>
<td>9.73</td>
<td>1.25</td>
<td>2.83</td>
</tr>
<tr>
<td>3</td>
<td>6.98</td>
<td>6.90</td>
<td>4.18</td>
<td>4.05</td>
</tr>
<tr>
<td>4</td>
<td>7.31</td>
<td>7.07</td>
<td>3.82</td>
<td>3.62</td>
</tr>
<tr>
<td>5</td>
<td>5.38</td>
<td>3.24</td>
<td>3.19</td>
<td>1.78</td>
</tr>
<tr>
<td>6</td>
<td>6.30</td>
<td>4.41</td>
<td>3.79</td>
<td>2.14</td>
</tr>
<tr>
<td>7</td>
<td>2.50</td>
<td>1.82</td>
<td>2.57</td>
<td>1.38</td>
</tr>
<tr>
<td>8</td>
<td>5.14</td>
<td>5.99</td>
<td>1.57</td>
<td>2.76</td>
</tr>
<tr>
<td>9</td>
<td>2.53</td>
<td>1.90</td>
<td>1.93</td>
<td>1.19</td>
</tr>
<tr>
<td>10</td>
<td>7.29</td>
<td>7.32</td>
<td>2.38</td>
<td>2.85</td>
</tr>
<tr>
<td>11</td>
<td>7.78</td>
<td>3.98</td>
<td>2.89</td>
<td>2.57</td>
</tr>
<tr>
<td>12</td>
<td>8.91</td>
<td>8.81</td>
<td>3.85</td>
<td>3.17</td>
</tr>
<tr>
<td>13</td>
<td>6.12</td>
<td>6.90</td>
<td>3.43</td>
<td>3.23</td>
</tr>
<tr>
<td>14</td>
<td>6.54</td>
<td>6.07</td>
<td>1.97</td>
<td>1.89</td>
</tr>
<tr>
<td>15</td>
<td>7.27</td>
<td>7.64</td>
<td>3.03</td>
<td>3.29</td>
</tr>
<tr>
<td>16</td>
<td>4.56</td>
<td>5.49</td>
<td>3.29</td>
<td>4.00</td>
</tr>
<tr>
<td>17</td>
<td>6.07</td>
<td>6.07</td>
<td>3.47</td>
<td>3.77</td>
</tr>
<tr>
<td>AVG</td>
<td>6.15</td>
<td>5.79</td>
<td>2.87</td>
<td>2.79</td>
</tr>
<tr>
<td>GLOBAL</td>
<td>5.97</td>
<td>3.60</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table C.3: Group Statistics per strategy for the bandwidth and coverage parameter.

<table>
<thead>
<tr>
<th>Method</th>
<th>N</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Std. Error Mean.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant density</td>
<td>204</td>
<td>.0188</td>
<td>.00971</td>
<td>.00068</td>
</tr>
<tr>
<td>Map coverage</td>
<td>204</td>
<td>.0274</td>
<td>.01514</td>
<td>.00106</td>
</tr>
<tr>
<td>Coverage</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constant density</td>
<td>204</td>
<td>6.1518</td>
<td>3.57253</td>
<td>.25013</td>
</tr>
<tr>
<td>Map coverage</td>
<td>204</td>
<td>5.7935</td>
<td>3.62634</td>
<td>.25839</td>
</tr>
</tbody>
</table>

Table C.4: Independent Samples t-test on the bandwidth and coverage parameters, grouped by strategy.
APPENDIX C. USER STUDY DATA

<table>
<thead>
<tr>
<th>Method</th>
<th>N</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Std. Error Mean.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
<td>Context</td>
<td>204</td>
<td>.0244</td>
<td>.01414</td>
</tr>
<tr>
<td>Coverage</td>
<td>No Context</td>
<td>204</td>
<td>.0218</td>
<td>.01256</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>No Context</td>
<td>204</td>
<td>7.0846</td>
<td>3.60607</td>
</tr>
<tr>
<td>Coverage</td>
<td>No Context</td>
<td>204</td>
<td>4.8607</td>
<td>3.23869</td>
</tr>
</tbody>
</table>

Table C.5: Group Statistics per context class for the bandwidth and coverage parameter.

<table>
<thead>
<tr>
<th>Method</th>
<th>Levene’s Test</th>
<th>t-test for Equality of Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
<td>F</td>
<td>Sig.</td>
</tr>
<tr>
<td>Equal variances assumed</td>
<td>2.342</td>
<td>.127</td>
</tr>
<tr>
<td>Equal variances not assumed</td>
<td>1.934</td>
<td>400.453</td>
</tr>
<tr>
<td>Coverage</td>
<td>6.338</td>
<td>.012</td>
</tr>
<tr>
<td>Equal variances assumed</td>
<td>6.553</td>
<td>401.401</td>
</tr>
<tr>
<td>Equal variances not assumed</td>
<td>6.553</td>
<td>401.401</td>
</tr>
</tbody>
</table>

Table C.6: Independent Samples t-test on the bandwidth and coverage parameters, grouped by context class.

C.2 Maps generated using mean and standard deviation

Below we show concrete examples of maps generated using the mean and standard deviation, as from the result of the user test, as parameters. For each test case, we show the reference image alongside the generated maps using both the constant density and the map coverage strategy.

<table>
<thead>
<tr>
<th>Test case</th>
<th>Constant Density</th>
<th>Map Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bandwidth</td>
<td>coverage</td>
</tr>
<tr>
<td>0</td>
<td>0.017</td>
<td>7.653%</td>
</tr>
<tr>
<td>1</td>
<td>0.024</td>
<td>7.858%</td>
</tr>
<tr>
<td>2</td>
<td>0.019</td>
<td>5.829%</td>
</tr>
<tr>
<td>3</td>
<td>0.017</td>
<td>9.594%</td>
</tr>
<tr>
<td>4</td>
<td>0.021</td>
<td>6.999%</td>
</tr>
<tr>
<td>5</td>
<td>0.020</td>
<td>5.276%</td>
</tr>
<tr>
<td>6</td>
<td>0.015</td>
<td>7.769%</td>
</tr>
<tr>
<td>7</td>
<td>0.017</td>
<td>3.864%</td>
</tr>
<tr>
<td>8</td>
<td>0.017</td>
<td>5.351%</td>
</tr>
<tr>
<td>9</td>
<td>0.020</td>
<td>3.741%</td>
</tr>
<tr>
<td>10</td>
<td>0.022</td>
<td>5.209%</td>
</tr>
<tr>
<td>11</td>
<td>0.016</td>
<td>4.676%</td>
</tr>
</tbody>
</table>

Table C.7: Mean preferred bandwidth and mean preferred map coverage per test.
C.2. MAPS GENERATED USING MEAN AND STANDARD DEVIATION

Figure C.1: Results using the mean bandwidth and coverage for test case 1 (see Table C.7): (a) reference image, (b) generated using constant density strategy, and (c) generated using map coverage strategy.

Figure C.2: Results using the mean bandwidth and coverage for test case 2 (see Table C.7): (a) reference image, (b) generated using constant density strategy, and (c) generated using map coverage strategy.

Figure C.3: Results using the mean bandwidth and coverage for test case 3 (see Table C.7): (a) reference image, (b) generated using constant density strategy, and (c) generated using map coverage strategy.
Figure C.4: Results using the mean bandwidth and coverage for test case 4 (see Table C.7): (a) reference image, (b) generated using constant density strategy, and (c) generated using map coverage strategy.

Figure C.5: Results using the mean bandwidth and coverage for test case 5 (see Table C.7): (a) reference image, (b) generated using constant density strategy, and (c) generated using map coverage strategy.

Figure C.6: Results using the mean bandwidth and coverage for test case 6 (see Table C.7): (a) reference image, (b) generated using constant density strategy, and (c) generated using map coverage strategy.
C.2. MAPS GENERATED USING MEAN AND STANDARD DEVIATION

Figure C.7: Results using the mean bandwidth and coverage for test case 7 (see Table C.7): (a) reference image, (b) generated using constant density strategy, and (c) generated using map coverage strategy.

Figure C.8: Results using the mean bandwidth and coverage for test case 8 (see Table C.7): (a) reference image, (b) generated using constant density strategy, and (c) generated using map coverage strategy.

Figure C.9: Results using the mean bandwidth and coverage for test case 9 (see Table C.7): (a) reference image, (b) generated using constant density strategy, and (c) generated using map coverage strategy.

Figure C.10: Results using the mean bandwidth and coverage for test case 10 (see Table C.7): (a) reference image, (b) generated using constant density strategy, and (c) generated using map coverage strategy.
Figure C.11: Results using the mean bandwidth and coverage for test case 11 (see Table C.7): (a) reference image, (b) generated using constant density strategy, and (c) generated using map coverage strategy.

Figure C.12: Results using the mean bandwidth and coverage for test case 12 (see Table C.7): (a) reference image, (b) generated using constant density strategy, and (c) generated using map coverage strategy.

Figure C.13: Range of possibilities: (a) minimum bandwidth and minimum coverage, (b) maximum bandwidth and minimum coverage, (c) center of the grid, (d) minimum bandwidth and maximum coverage, and (e) maximum bandwidth maximum coverage.
Appendix D

Derivations

D.1 Locating min- and max-points

The goal is to find the minimal and maximal values (after bilinear interpolation), within the density map \( \mathcal{F} = (f, G) \) (where \( G \) is the grid), for points that are on the edge \( e \) in the Delaunay triangulation \( T \). We look at the intersections of \( e \) with each quad \( Q \), defined by the grid points of \( G \). This intersection is the line-segment \( l \). Originally, to compute the values at non grid points in the density map, we use bilinear interpolation.

We first give the function to compute the interpolated value at a point \( p = (x,y) \) (see Equation (D.1)), and then use the derivative of this function (Equation (D.3)) to find the minimal and maximal points. Standard bilinear interpolation is done by first interpolating in the \( x \)-direction. This results in two values; \( v_1 \), interpolated between \( f_{00} \) and \( f_{10} \); and \( v_2 \), interpolated between \( f_{01} \) and \( f_{11} \). Next, to get the final interpolated value \( f(x,y) \), we interpolate in the \( y \)-direction between \( v_1 \) and \( v_2 \). To simplify notation, we use the coordinate system of quad \( Q \), i.e., the endpoints of \( Q \) are at \((0,0)\), \((1,0)\), \((0,1)\), \((1,1)\).

\[
\begin{align*}
v_1 &= (1 - x) \cdot f_{00} + x \cdot f_{10} \\
v_2 &= (1 - x) \cdot f_{01} + x \cdot f_{11} \\
f(x,y) &= (1 - y) \cdot v_1 + y \cdot v_2, \text{ substitute and simplify} \\
f(x,y) &= f_{00} \cdot (1 - x) \cdot (1 - y) + f_{10} \cdot x \cdot (1 - y) \\
&\quad+ f_{01} \cdot (1 - x) \cdot y + f_{11} \cdot x \cdot y \\
&= \left[1 - x \right] \times \begin{bmatrix} f_{00} & f_{01} \\ f_{10} & f_{11} \end{bmatrix} \times \begin{bmatrix} 1 - y \\ y \end{bmatrix},
\end{align*}
\tag{D.1}
\]

where we use \( f_{00}, f_{10}, f_{01} \) and \( f_{11} \) to denote the scalar values at the four endpoints of \( Q \). These values are known, as they can be taken from the density map \( \mathcal{F} \).

We parameterize the line-segment \( l \), by specifying each point on \( l \) as \( (x_0, y_0) + t \cdot (\Delta_x, \Delta_y) \) for \( t \in [0,1] \), where \((x_0, y_0)\) is an arbitrary endpoint of \( l \) and \((\Delta_x, \Delta_y)\). We can now express the value of \( f \) for any point on \( l \) as a function of \( t \):

\[
f_l(t) = f(x_0 + t\Delta_x, y_0 + t\Delta_y).
\tag{D.2}
\]
We compute the derivative \( \frac{d}{dt} f_l(t) \) of \( f_l \):

\[
\frac{d}{dt} f_l(t) = f_{00} \cdot (-1 \cdot t \Delta x - x_0) \cdot \Delta y - \Delta x \cdot (1 - t \Delta y - y_0)) + \\
f_{01} \cdot ((1 - t \Delta x - x_0) \cdot \Delta y - \Delta x \cdot (t \Delta y + y_0)) + \\
f_{10} \cdot (-t \Delta x + x_0) \cdot \Delta y + \Delta x \cdot (1 - t \Delta y - y_0)) + \\
f_{11} \cdot ((t \Delta x + x_0) \cdot \Delta y + \Delta x \cdot (t \Delta y + y_0))
\]

(D.3)

Solving \( \frac{d}{dt} f_l(t) = 0 \) we get a solution for \( t \) that maximizes or minimizes \( f_l(t) \):

\[
t = (f_{00} \cdot (\Delta x + \Delta y - x_0 \cdot \Delta y - \Delta x \cdot y_0) + \\
f_{10} \cdot (-\Delta x + x_0 \cdot \Delta y + \Delta x \cdot y_0) + \\
f_{01} \cdot (-\Delta y + x_0 \cdot \Delta y + \Delta x \cdot y_0) + \\
f_{11} \cdot (-\Delta x \cdot y_0 - x_0 \cdot \Delta y))
\]

(D.4)

\[
\frac{1}{2 \cdot (f_{00} - f_{10} - f_{01} + f_{11}) \cdot \Delta x \cdot \Delta y}
\]

Due to the bilinear interpolation, we cannot have a maximum and a minimum on the same line-segment \( l \). The denominator of the fraction \( \frac{1}{2 \cdot (f_{00} - f_{10} - f_{01} + f_{11}) \cdot \Delta x \cdot \Delta y} \) indicates the special cases where \( f_{00} - f_{01} = f_{10} - f_{11} \) (opposing edges in \( Q \) have the same gradient, which means the interpolated value are exactly placed on a plane), \( \Delta x = 0 \) (\( l \) is vertical) or \( \Delta y = 0 \) (\( l \) is horizontal). The fourth case is that \( t \) is not within the range \([0..1]\). In all four cases, we note that the maximal and minimal values can be found at the endpoints of \( l \).