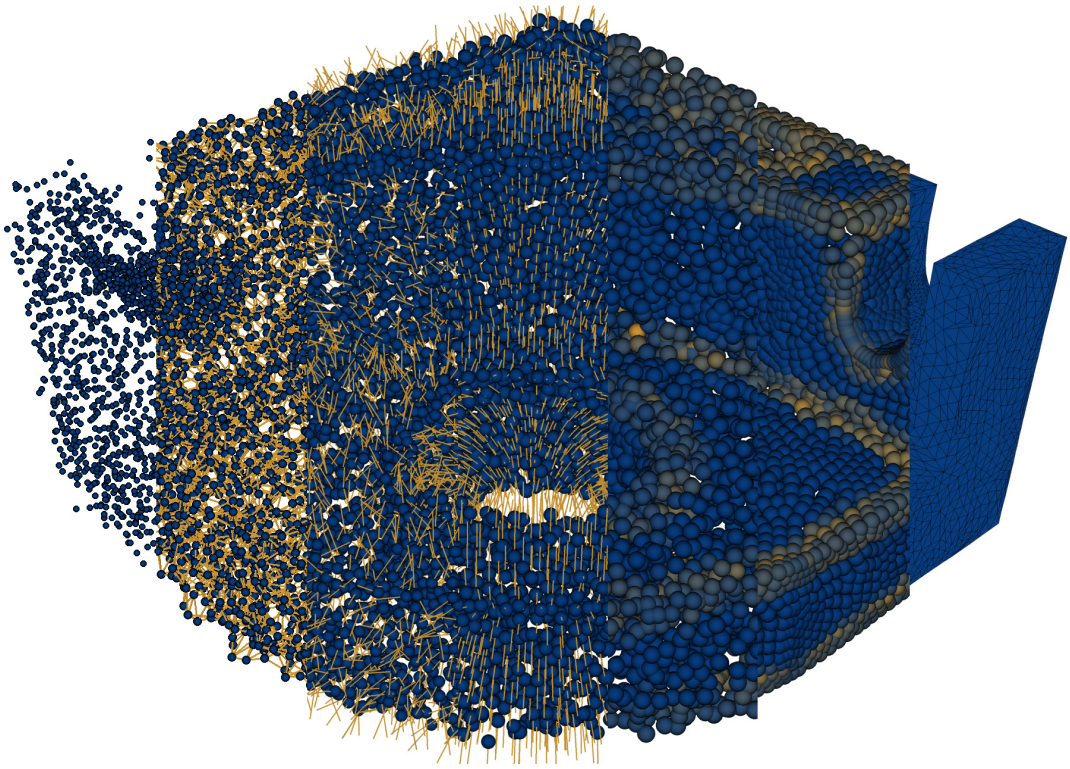
Neighborhood Data Structures, Manifold Properties, and Processing of Point Set Surfaces



Martin Skrodzki, M.Sc.

Dissertation zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.),
eingereicht am

Fachbereich Mathematik und Informatik der Freien Universität Berlin

Berlin, 2019

Erstgutachter und Betreuer:	Prof. Dr. Konrad Polthier	(Freie Universität Berlin)
Zweitgutachter:	Prof. David Levin	(Tel Aviv University)

Datum der Disputation: 3. Juli 2019

Table of Contents

Introduction

Starting from its first presentation at the E3 fair in 2009, the Microsoft Kinect sensor introduced geometry processing to living rooms worldwide. In real-time, the scanned bodies of players are processed, joints are recognized and animated models are rendered according to the motions of the users. In Apple's iPhones, infrared lights project a pattern onto the face of the user. A corresponding camera reads the reflection and thereby creates a 3-dimensional fingerprint of the user's face which is in turn used to unlock the device. The whole procedure is performed within milliseconds. While the Kinect sensor and the iPhone acquire geometrical data to be used, 3D printers create solid manifestations of such data. The corresponding techniques are available since the 1980s and have even transcended earth, as 3D printing is performed on the international space station, see [Joh+14].

For both of these and more applications, geometric models have to be represented on a multitude of devices. Thus, real-world models have to be discretized for digital storage. Here, one can distinguish between the following three scenarios.

1. The discrete geometry is represented as a simplicial mesh, the most prominent instances being triangulated surfaces or tetrahedralized volumes embedded in $\mathbb{R}^3$. The motivation for this representation comes both from a classical mathematical perspective (for instance topological results can often be computed using the purely combinatorial nature of a triangulation) as well as from a computer graphics perspective since it is these primitives that can be efficiently rendered by the established graphics pipeline. Consequently, there is a rich theory of discretized concepts from differential geometry such as discretized Laplace operators, curvature flows, or parametrization methods, to name just a few.

2. The discrete geometry is represented by a higher-order spline model. This representation is mostly motivated from modern modeling applications used in industrial design and computer graphics. However, due to the more complicated nature of each cell in this representation, the discretization of differential geometric concepts on these geometries turns out to be much more involved. A better understanding e.g. of appropriate test functions and finite-element formulations on such spline surfaces is a vivid research topic in the isogeometric analysis community.

3. The discrete geometry is a raw point set, without any connectivity information. This representation is motivated by the nowadays wide-spread availability of 3D scanning devices, both in the professional market as well as in the entertainment industry. These devices deliver point sets as a most natural representation of the scanned objects. However, the theory for possible discretizations of differential geometric concepts is still underdeveloped in comparison with simplicial or spline representations.

It is these mesh-less geometries we would like to focus our research on. They have a long history in geometry processing and computer graphics as they naturally arise in 3D acquisition processes. A guiding principle of these algorithms is the direct processing of raw scanning data without prior meshing—a principle that is well-established in classical numerical computations. However, the

usage of these computations mostly restricts to full dimensional domains embedded in $\mathbb{R}^2$ or $\mathbb{R}^3$, where point sets usually represent surfaces and therefore lower-dimensional domains. A thorough investigation of a differential geometric representation of point set surfaces and their properties is not available.

Points have been introduced in 1985 as display primitives for computer graphics by the authors of [LW85]. There has been substantial research progress in the years 2004–2007 as presented at the Symposia on Point Based Graphics, see [Gro+04; Ale+05; Bot+06; Bot+07]. Despite the diverse applications of point sets, e.g. in face recognition [BF05], traffic accident analysis [Buc+07], or archeology [Lev+00], the field still lacks mathematically sound theory, in particular for the newer developments in this area.

The use of mesh-less methods has several direct implications: Computation is performed on the point set and does not require any preprocessing step for meshing. Therefore, data can be processed immediately after acquisition, cf. [Lin01]. Since no mesh is computed, also, one does not have to store the edges and faces, resulting in a more memory-efficient setup. However, there are also disadvantages. Point sets do neither provide explicit connectivity information nor do they include faces for the use as area weighting terms or for easy rendering. To overcome some of these disadvantages while retaining the advantages is the aim of this thesis.

Structure of the Thesis and Summary of Main Contributions

The thesis covers three topics all centered in the context of point set processing. In the following, we will shortly present each topic with its motivation, explain the performed research, and highlight the contributions. In case the presented results have been published prior to the publication of this thesis, the corresponding reference is given.

Notions of Neighborhood and corresponding Data Structures

The first topic concerns notions of neighborhood and corresponding data structures. Many researchers have recognized the importance of high-quality neighborhood relations. For example, the authors of [LP05] report that the results of their anisotropic smoothing algorithm heavily depend on the neighborhood structure imposed on the point set. Despite their advantages in storage space and easy acquisition, the missing neighborhood relation is a significant downside to point set representations. The purpose of this first part is to discuss neighborhood concepts as well as a data structure for fast single-core and fast parallelized computation respectively.

Neighborhoods in Point Sets Usual approaches to point set neighborhoods include combinatorial k nearest neighbors or geometric neighborhoods. However, both do not include curvature or normal information of the point set. Following an idea presented in [YRP17], in Section 1, we formulate the concept of neighborhoods that do not aim for a heuristically chosen size or try to obtain an optimal size by a specialized error measure. Our neighborhoods rather adapt to the shape of the geometry. The presented approach is evaluated experimentally with error measures derived in the work of [WJM14]. In experiments on both CAD and real-world models, we establish that incorporating shape-aware weights into the computation yields smaller error measures than the uniform weights from [WJM14] or the sharp cut-off neighborhoods used in [Yad+18a]. Furthermore, we improve the denoising results of [Yad+18a] and incorporate our neighborhood concept in the Moving Least Squares (MLS) procedure of [SL16]. Here, we see that it delivers results comparable to those of RIMLS, see [ÖGG09].

k-d Trees In any practical application, the fast determination of neighborhoods is of high importance. The most prominent choice for a data structure is—despite its age—a k-d tree. It has been presented in 1977, with the groundbreaking result of a very low runtime for a neighborhood finding algorithm. Namely, for n points, the authors of [FBF77] prove that a nearest neighbor query can be answered in an expected time of $\Theta O(\log(n))$. Even though this key result is used far and wide, no modern formulation of the original proof is available. Therefore, Section 2 presents a modern and elaborate version of the original proof.

The Neighborhood Grid Despite the praise given to k-d trees in the previous paragraph, they have a significant downside. Namely, they are not designed for parallelization. As workstations nowadays can make use of graphics cards to speed up computations, it is necessary to also investigate the field of neighborhood computing data structures to benefit from parallelization. A corresponding structure—the neighborhood grid—was presented by the authors of [Jos+09; Jos+15] and further investigated by [MW15]. Nonetheless, many open questions concerning both combinatorial properties of the data structure and the quality of its results remain. In Section 3, the thesis answers several of them, namely:

- Give a proof of asymptotic time-optimality of a building algorithm.
- Comparison of the single-core building algorithm with the parallel building algorithm of Malheiros and Walter, cf. [MW15].
- Combinatorial results on the number of states of the data structure.
- Analysis of the neighborhood quality obtained from the data structure.

The results of this research are available on ArXiv, see [SRP17], and have been presented at the EuroCG18 conference, see [Skr+18].

Manifold Structure for Point Set Surfaces

The second main topic of this thesis deals with manifold structures for point set surfaces. From a significant amount of real-world objects, while 3D-scanning them, only the surface is acquired for further processing in CAD or other applications. When the surface of the underlying real-world geometry has the structure of a manifold, it can be expected that this structure is reflected by any point set acquired from the geometry. Even when the faces of the geometry are smooth manifold patches, there is no theory available in the setting of point sets to reflect their manifold properties.

Manifold Theory and Formulations for Point Set Manifolds By definition, each point of a point set is itself a 0-dimensional object. Naturally, they could be seen as a 0-manifold. In Section 4, we establish that this is not a meaningful choice in the given application process. Then, we proceed to construct a scheme to handle point sets acquired from surfaces as 2-dimensional representations of a 3D object's surface by considering a transition manifold. This transition manifold is provided e.g. by the MLS procedure of [SL16].

Variational Shape Approximation Having established a manifold scheme in Section 4, the crucial question is how to define charts on a point set. The aforementioned MLS procedure results in highly localized charts. That is, each point of the point set is assigned its own chart, which results in a large number of transition maps to be computed. A completely different approach is taken by [Li+11] who aim at a global parametrization of a given point set by generalizing

a technique presented by [KNP07] and [BZK09] for meshes. This in turn gives one very large representation. We aim at a solution between these two extremes. To achieve this, in Section 5, we turn to an algorithm of [CAD04]. Similar to [LB16], we translate it to the setting of point sets. It generates charts which each incorporate regions of similar normal behavior. We further enrich the algorithm by a *split* and *merge* procedure to become independent of a prescribed number of charts which comes from the underlying algorithm of [Llo82]. Finally, we provide several numerical examples concerning its performance.

Robust and Efficient Processing of Point Sets

Third and finally, algorithms have to work efficiently and robustly on the point set. While meshed geometries provide an intuitive and natural weighting by the areas of the faces, point sets can at most work with distances between the points. This introduces a new level of difficulty to be overcome by any point set processing algorithm.

Directional Density Measure to Intrinsically Estimate and Counteract Non-uniformity in Point Sets When considering point set samples, many algorithms make the explicit or implicit assumption of a uniform sampling. However, many acquisition processes do not produce such results, but provide rather non-uniform representations. In order to counteract this non-uniformity, we introduce new weights in Section 6. These are based on a discrete directional density measure for point sets and can be computed intrinsically from the point set without additional information. Our evaluation within two discrete differential operators showcases the benefits of our technique. This research has been published in [SJP18].

Feature Detection from Moving Least Squares A frequent problem in the processing of geometries is the detection of features like corners or edges. For example, in the context of denoising—see next paragraph—features should be retained while removing noise that was additionally added during acquisition. In addition to many feature detection algorithms present, we aim at a procedure which mathematically guarantees features of a certain size to be detected. For this, in Section 7, we turn to the MLS method, see [SL16]. First, we derive the necessary theory to prove that the MLS approach will detect features of a certain size. Second, from several different feature quantities derived from MLS, we identify four which can be used to detect features. These are compared experimentally.

Constraint-Based Point Set Denoising using the Normal Voting Tensor and Restricted Quadratic Error Metrics As mentioned above, additional noise added to the point set during the acquisition process is problematic for several steps of the geometry processing pipeline. Thus, reliable denoising algorithms are necessary to remove noise components. These still have to retain the features of the geometry. The authors of [YRP17] present an algorithm for denoising of meshes using the *element-based normal voting tensor*. Following this idea, we derive a normal voting tensor based on point sets and thus generalize the concept to the mesh-free setting. To assure feature preservation in particular at the corners, we include quadratic error metrics following [GH97]. The result is a robust iterative smoothing algorithm which has been published in [Yad+18a].

For an overview on the notation used in this thesis, see Appendix A. Possibly ambiguous notation is also defined there.

Publications prior to the Thesis

Several parts of this thesis have been published as journal articles or in conference proceedings as indicated in the paragraphs above. The work on k-d trees is related to results of a Master thesis, see [Skr14a]. It led to a publication in a series of articles at the Bridges conference, refer to [SRP16; SP17; SP18]. Works in other fields, published or handed in prior to this thesis, are: [DVS18; Skr19].

List of Publications prior to the thesis

[DVS18] Milena Damrau, Hernán Villamizar, and Martin Skrodzki. "Eine Datenanalyse der Persistenz und Leistung von Schulkindern im Wettbewerb "Mathe im Advent"". In: *Beiträge zum Mathematikunterricht 2018*. Münster: WTM Verlag für wissenschaftliche Texte und Medien, 2018, pp. 421–424.

[SJP18] Martin Skrodzki, Johanna Jansen, and Konrad Polthier. "Directional density measure to intrinsically estimate and counteract non-uniformity in point clouds". In: *Computer Aided Geometric Design* (2018). ISSN: 0167-8396.

[Skr+18] Martin Skrodzki et al. "Combinatorial and Asymptotical Results on the Neighborhood Grid Data Structure". In: *EuroCG18 Extended Abstracts*. 2018, 30:1–30:6.

[Skr14a] Martin Skrodzki. "Neighborhood Computation of Point Set Surfaces". MA thesis. Freie Universität Berlin, 2014.

[Skr19] Martin Skrodzki. "Einfach erstaunlich schwierig: Vom Staunen in der Mathematik". In: *Staunen. Perspektiven eines Phänomens zwischen Natur und Kultur*. Fink, 2019.

[SP17] Martin Skrodzki and Konrad Polthier. "Turing-Like Patterns Revisited: A Peek Into The Third Dimension". In: *Proceedings of Bridges 2017: Mathematics, Art, Music, Architecture, Education, Culture*. Ed. by David Swart, Carlo H. Séquin, and Kristóf Fenyvesi. Phoenix, Arizona: Tessellations Publishing, 2017, pp. 415–418. ISBN: 978-1-938664-22-9.

[SP18] Martin Skrodzki and Konrad Polthier. "Mondrian Revisited: A Peek Into The Third Dimension". In: *Proceedings of Bridges 2018: Mathematics, Art, Music, Architecture, Education, Culture*. Ed. by Eve Torrence et al. Phoenix, Arizona: Tessellations Publishing, 2018, pp. 99–106. ISBN: 978-1-938664-27-4.

[SRP16] Martin Skrodzki, Ulrich Reitebuch, and Konrad Polthier. "Chladni Figures Revisited: A Peek Into The Third Dimension". In: *Proceedings of Bridges 2016: Mathematics, Music, Art, Architecture, Education, Culture*. Ed. by Eve Torrence et al. Phoenix, Arizona: Tessellations Publishing, 2016, pp. 481–484. ISBN: 978-1-938664-19-9.

[SRP17] Martin Skrodzki, Ulrich Reitebuch, and Konrad Polthier. "Combinatorial and Asymptotical Results on the Neighborhood Grid". In: *ArXiv e-prints* (Oct. 2017).

[Yad+18a] Sunil Kumar Yadav et al. "Constraint-based point set denoising using normal voting tensor and restricted quadratic error metrics". In: *Computers & Graphics* 74 (2018), pp. 234–243. ISSN: 0097-8493.

Acknowledgment

This thesis has been supported both financially and ideally by several parties. I would like to thank both the Einstein Center for Mathematics (ECMath) in Berlin and the German National Academic Foundation for bestowing a scholarship on me which made the writing of this thesis possible. By the generous support of Freie Universität Berlin and Tel Aviv University, I was able to attend a joint workshop with Tel Aviv University in 2016 where I got to know Prof. Levin. The Minerva foundation sponsored a short-term research grant by which I was able to travel to Tel Aviv University in 2017 which sparked a fruitful collaboration on the MLS procedure. Throughout the process of writing, I have been supported by several seminars organized by both my graduate schools, the Berlin Mathematical School (BMS) and the Dahlem Research School (DRS). I acknowledge the support by the DFG Collaborative Research Center TRR 109, 'Discretization in Geometry and Dynamics', within whose project C05 most of the presented research was conducted. Furthermore, the Society of Industrial and Applied Mathematics (SIAM) sponsored two travel grants which enabled me to visit two conferences in 2017 and 2019 respectively. Finally, I wish to thank AIM@SHAPE and Stanford 3D scanning repository[1] for providing several datasets and the Gemeentemuseum Den Haag for allowing me to print "Tableau I" by Piet Mondrian in this thesis.

[1]Find the repositories at http://visionair.ge.imati.cnr.it/ and http://graphics.stanford.edu/data/3Dscanrep/ respectively.

I Notions of Neighborhood and corresponding Data Structures

1 Neighborhoods in Point Sets

Despite their versatility and their advantages—like low storage costs—point sets have a significant downside to them when compared with mesh representations: They are not equipped with connectivity information. This is mostly due to the acquisition process. Consider for example a manually guided scanning device. The operator will scan those areas of the real-world objects multiple times that have very sharp features. Thus, occlusion is prevented and the whole geometry is captured. Even though each scan can provide connectivity information on the respectively acquired points, the complete point set P obtained via registration of the individual scans (see e.g. [Bel+14]) does not provide global connectivity information in general. Thus, the notion of neighborhoods has to be defined and computed for each point $p \in P$.

In particular in the context of geometry processing, high quality neighborhood relations on the processed point sets are of great importance. For example, in the context of anisotropic smoothing of noisy point sets, the authors of [LP05] remark on the relevance of point set neighborhoods. They find that the quality of the smoothed point set is drastically amplified when using neighborhoods as close as possible to those of the originally sampled object. However, the authors do not provide an algorithm to produce these neighborhoods.

This section presents different notions of neighborhoods: combinatorial k nearest neighbors (Section 1.1.1), metric neighborhoods (Section 1.1.2), a combination of both (Section 1.1.3), and the concept of relaxed neighborhoods (Section 1.1.4). A brief discussion on related work concerning optimal neighborhood sizes (Section 1.2) settles the ground for the main contributions of this section:

- Definition of anisotropic neighborhood selection for shape-aware computations on point sets (Section 1.3).
- Experimental evaluation of the anisotropic neighborhood selection on both a variety of models (Section 1.4) and within applications (Section 1.5).

The results of this section are currently under review for the SPM'2019 conference, see list of publications prior to the thesis, page 5.

1.1 Neighborhood Concepts

1.1.1 k Nearest Neighbors

The widest used definition of neighborhood in the context of point sets is the k *nearest neighborhood*. For a point set $P = \{p_i \in \mathbb{R}^d \mid i = 1, \dots, n\}$, a point $p_i \in P$, a distance mea-

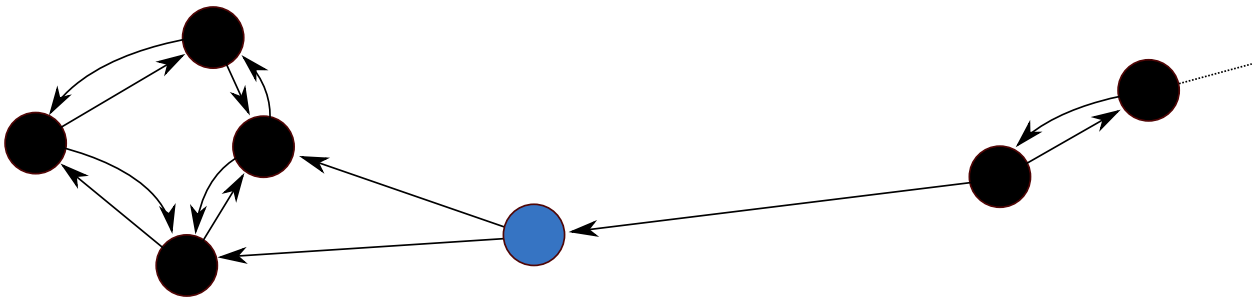


Figure 1.1: *Several points are shown in $\mathbb{R}^2$. The arrows indicate the corresponding $k = 2$ nearest neighbor graph resulting from (1.2) which clearly has non-symmetric edges. Note how the blue point in the center favors neighbors in the dense left region instead of favoring a neighborhood which is distributed as uniformly as possible around the blue point. See Section 6 for a thorough discussion for non-uniformity in point sets.*

sure[1] $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, and a number $k \in [n-1]$ it is defined to be

$$\begin{aligned}\mathcal{N}_k(p_i) = \{&p_{i_1}, \ldots, p_{i_k} \in P \backslash \{p_i\} \\ &\text{s.t. } d(p_i, p_{i_k}) \leq d(p_i, q)\ \forall q \in P \backslash \{p_i, p_{i_1}, \ldots, p_{i_k}\} \text{ and} \\ &d(p_i, p_{i_1}) \leq \ldots \leq d(p_i, p_{i_k}),\ i_1, \ldots, i_k \in [n], i_\ell \neq i_m\ \forall \ell, m \in [k]\}.\end{aligned} \tag{1.1}$$

It is a strictly combinatorial notion as any point p_i is given exactly k neighbors, no matter how far those are from p_i in the given distance measure. From the $\mathcal{N}_k(p_i)$ one can define the k *nearest neighbor graph* by considering vertices $V = \{p_i \mid i \in [n]\}$ and a set of directed edges given by

$$E = \{(p_i, p_j) \mid p_j \in \mathcal{N}_k(p_i)\}. \tag{1.2}$$

Note that the edges of this graph are directed. That is, the neighborhood relation defined in Equation (1.1) is not symmetric. A simple example is given in Figure 1.1. Furthermore, the k nearest neighborhood favors densely sampled regions resulting in biased neighborhoods, as depicted in Figure 1.1.

Finally, it is a priori unclear what is a good and suitable choice for the value k in a given application. It is well-known that the average valence of a triangle mesh is 6. This can be derived from Euler's formula

$$v - e + f = \chi \tag{1.3}$$

with v, e, and f the number of vertices, edges, and faces of the mesh respectively and χ the Euler characteristic of the geometry. For sufficiently large meshes of sufficiently small genus and no present boundary, χ is negligible and $2e = 3f$, so that $e/v = 3$ and since every edge is shared by two vertices, the average vertex has valence 6. Therefore, typically k is small ($k < 20$), but in particular for varying densities in the point set, it might be important to vary k throughout the process and even to determine a specific k_i for every point $p_i \in P$.

1.1.2 Metric Balls

A different approach to defining neighborhoods, as opposed to the k nearest neighborhood of Section 1.1.1, is utilizing metric balls. Given a point set $P = \{p_i \in \mathbb{R}^d \mid i = 1, \ldots, n\}$, $p_i \in P$,

[1] While the theory holds for any distance measure, i.e. any metric, in the following we will use the Euclidean metric.

and a value $\varepsilon \in \mathbb{R}_{\geq 0}$ the ε *nearest neighborhood* is defined to be

$$\mathcal{N}_\varepsilon(p_i) = (P \backslash \{p_i\}) \cap B_\varepsilon(p_i), \tag{1.4}$$

where $B_\varepsilon(p_i) = \{p \in \mathbb{R}^3 \mid d(p, p_i) \leq \varepsilon\}$ for some distance measure[2] d. Compared to the combinatorial definition of Equation (1.1), this neighborhood definition preserves more metric information. However, for a given $\varepsilon \in \mathbb{R}_{\geq 0}$ and $p_i \in P$, the set $\mathcal{N}_\varepsilon(p_i)$ might be empty or could contain the whole set P. As in Section 1.1.1, the neighborhood can be used to define the ε *nearest neighbor graph* by $V = \{p_i \mid i \in [n]\}$ and a set of edges given by

$$E = \{(p_i, p_j) \mid p_i \in N_\varepsilon(p_j)\}. \tag{1.5}$$

Note that the edges of this graph are undirected, since the neighborhood relation implied by (1.4) is symmetric, as long as the same ε is used for all $\mathcal{N}_\varepsilon(p_i)$. In [FR01, p. 8], the authors discuss the advantage of symmetric neighborhoods in the context of surface reconstruction utilizing radial basis functions. Although the ε nearest neighborhood might still favor densely sampled regions, by adjusting ε, this issue can be rectified easier than for k nearest neighborhoods. Still, it is unclear what choice of ε is suitable for a given point set P and different applications. As in the combinatorial case, varying the value of ε over the point set might be beneficial.

1.1.3 Combination

Having defined k as well as ε nearest neighborhoods, a natural next step is to combine the two definitions. Namely, we define a k-ε *nearest neighborhood* by considering the intersection of both, i.e. for some point set $P = \{p_i \in \mathbb{R}^d \mid i \in [n]\}$, $p_i \in P$, $k \in [n-1]$, $\varepsilon \in \mathbb{R}_{\geq 0}$ we define

$$\mathcal{N}_{k,\varepsilon}(p_i) = \mathcal{N}_k(p_i) \cap \mathcal{N}_\varepsilon(p_i). \tag{1.6}$$

Note that this does carry disadvantages from both definitions. The relation induced by Equation (1.6) is not symmetric. Although a point p_i now has at most k neighbors, the set $\mathcal{N}_{k,\varepsilon}$ could contain any number of points from 0 to k. Furthermore, the neighbors found are local, i.e. at most ε away from p_i. This type of neighborhood was successfully used e.g. in [LP05] and also studied in [Skr14b].

1.1.4 Relaxed Neighborhoods

As pointed out above, the definition of neighborhood used might depend on the specific application. In [Jan17], the data of interest is obtained via a LiDaR scanner on a driving car. The grid of points returned from the scanner is very regular with only slight disturbances. In order to keep the symmetry of the data present despite the small perturbations, Jansen proposes to introduce a relaxation range ε_r. Furthermore, as points too close to the original query point p_i introduce numerical errors, these are excluded using the machine accuracy ε_m. The *relaxed* versions of both the combinatorial neighborhood from Equation (1.1) and the metric neighborhood from Equation (1.4) are then given by

$$\begin{aligned} \tilde{\mathcal{N}}_k(p_i) = &(\mathcal{N}_k(p_i) \backslash \{p_j \in P \mid d(p_i, p_j) < \varepsilon_m\}) \\ &\cup \{p_j \in P \mid \operatorname{abs}(d(p_j, p_i) - \max_{p_\ell \in \mathcal{N}_k(p_i)} d(p_\ell, p_i)) \leq \varepsilon_r\}, \end{aligned} \tag{1.7}$$

[2] See Footnote 1, page 8.

$$\begin{aligned}\tilde{\mathcal{N}}_\varepsilon(p_i) = &(\mathcal{N}_\varepsilon(p_i) \backslash \{p_j \in P \mid d(p_i, p_j) < \varepsilon_m\}) \\ &\cup \{p_j \in P \mid \operatorname{abs}(d(p_j, p_i) - \max_{p_\ell \in \mathcal{N}_\varepsilon(p_i)} d(p_\ell, p_i)) \leq \varepsilon_r\},\end{aligned} \tag{1.8}$$

where $\operatorname{abs}(x)$ denotes the absolute value of $x \in \mathbb{R}$. Utilizing these relaxed versions of neighborhoods comes with all advantages and disadvantages of the original definitions. However, the additional advantage is the avoidance of numerical errors caused by too closely clustered points and the inclusion of points that miss the original neighborhood definition by a small margin ε_r. This definition of neighborhood was successfully applied in the context of discrete directional density measures, see Section 6.

1.2 Neighborhood Sizes

All neighborhood definitions presented above (1.1), (1.4), (1.6), (1.7), and (1.8) depend completely on the chosen values for the number k of neighbors or the radius ε of the neighborhood. Choosing these important values in an optimal manner is therefore an important research question.

1.2.1 Heuristics

The authors of [Ale+01] use a global radius ε as in Equation (1.4). They change it to affect the running time of their algorithm. In [PGK02], the authors fix a combinatorial number k of neighbors to be sought. Then, for each point p_i, these k neighbors are found, which fixes a radius ε_i to the farthest of them. Finally, the neighbors within radius $\varepsilon_i/3$ are used. Thus, their approach resembles the neighborhood from Equation (1.4).

The method used in [Pau+03] is more involved. The authors recognize that both a too large or too small radius ε leads to problems and thus aim for a local adaption like [PGK02]. A local density estimate δ_i around each point $p_i \in P$ is computed from the smallest ball centered at p_i, containing $\mathcal{N}_k(p_i)$, where k is found experimentally to be best chosen from $\{6, \ldots, 20\} \subset \mathbb{N}$. Given the radius ε_i of this ball, the local density is set to be $\delta_i = k/\varepsilon_i^2$. In a second step, a smooth density function δ is interpolated from the local density estimates δ_i.

In the context of surface reconstruction, the authors of [FR01] discuss several choices for neighborhoods and corresponding weights. While two of the three presented methods simply use geometric neighborhoods (1.4), the third method takes a different approach. Namely, the authors collect all neighbors of p_i in a "large" ball ([FR01, page 7]) around p_i. Then, they fit a plane to this preliminary neighborhood and project all neighbors and p_i onto this plane. On the projections, a Delaunay triangulation is built and the induced neighborhood of the triangulation is used in the following computations.

A completely different approach is taken by [BL12]. The authors first calculate features of a point set based on differently sized neighborhoods. Then, they use a training procedure to find the combination of neighborhood sizes that provides the best separation of different feature classes.

1.2.2 Neighborhood Sizes from Error Functionals

While the approaches presented above are based on heuristics, some works try to deduce an optimal k for the k nearest neighborhoods based on error functions. While investigating a method for nonparametric density estimation using the k nearest neighbor approach, the authors of [FH73] optimize the value for k according to a mean-square-error criterion. They find that the optimal k depends upon the dimension of the observation space and the underlying distribution of the the point set.

The authors of [LCL06] work in the context of the MLS framework (see [Ale+01; Ale+03; Lev98; Lev04; SL16]) for function approximation. The authors perform an extensive error analysis to quantify the approximation error both independent and depending on the given data. Finally, they obtain an error functional. This functional is then evaluated for different neighborhood sizes k. The neighborhood $\mathcal{N}_k$ yielding the smallest error is then chosen to be used in the actual MLS approximation.

In contrast, the authors of [MNG03] deduce an error bound on the normal estimation obtained from different neighborhood sizes. Utilizing this error functional, they obtain the best suited neighborhood size for normal computation.

The work of [LCL06] heavily depends on the MLS framework in which the error analysis is deduced, while the work of [MNG03] depends on the framework of normal computation. Both methods aim at finding an optimal neighborhood size k or ε. In the following, we will consider neighborhoods that are not derived by their size, but by their shape, i.e. guided by normal or curvature information. The first to mention such an idea were Hoppe et al. in their 1992 paper [Hop+92], where they asked for an adaptive neighborhood size k and proposed:

> *To select and adapt k, the algorithm could incrementally gather points while monitoring the changing eigenvalues of the covariance matrix.*

Following this idea, several authors take into account the covariance matrix at a point $p_i \in P$ given as

$$C_i := \sum_{p_j \in \mathcal{N}_k(p_i)} (p_j - \bar{p}_i)(p_j - \bar{p}_i)^T, \qquad C_i \in \mathbb{R}^{3\times 3} \tag{1.9}$$

where $\bar{p}_i = \frac{1}{k}\sum_{p_j \in \mathcal{N}_k(p_i)} p_j$ is the barycenter of the neighbors of p_i and v^T denotes the transpose of a vector $v \in \mathbb{R}^3$. The covariance matrix C_i is symmetric and positive-semi-definite. Thus, it has three non-negative eigenvalues, which in the following we will denote by $\lambda_{i,1} \geq \lambda_{i,2} \geq \lambda_{i,3} \geq 0$. Furthermore, we will denote their sum by $\lambda_i^\Sigma = \sum_{\ell=1}^{3} \lambda_{i,\ell}$.

In the work of [Pau+03], the authors grow a neighborhood and consider the surface variation[3]

$$C_i^\lambda = \frac{\lambda_{i,3}}{\lambda_i^\Sigma}$$

as a measure how large to consider a neighborhood around each point p_i. The same quantity C_i is used by [BL06]. However, they do not grow a neighborhood, but choose a size k for it according to a consistent curvature level.

The authors of [WJM14] take a more general approach in the context of segmentation of 3D point sets. They also use the concept of a combinatorial neighborhood (1.1), going back to results of [LP01]. They proceed to consider three more quantities derived from the eigenvalues of the covariance matrix. Namely, they work with linearity L_λ, planarity P_λ, and scattering S_λ given by[3]

$$L_i^\lambda = \frac{\lambda_{i,1} - \lambda_{i,2}}{\lambda_{i,1}}, \qquad P_i^\lambda = \frac{\lambda_{i,2} - \lambda_{i,3}}{\lambda_{i,1}}, \qquad S_i^\lambda = \frac{\lambda_{i,3}}{\lambda_{i,1}} \tag{1.10}$$

and representing 1D, 2D, and 3D features in the point set [Dem+11]. Finally, following the concept of entropy by Shannon [Sha48], the authors consider the measure[4]

$$E_i^{\dim} = -L_i^\lambda \ln(L_i^\lambda) - P_i^\lambda \ln(P_i^\lambda) - S_i^\lambda \ln(S_i^\lambda). \tag{1.11}$$

[3] See Section 1.3 for a discussion of the cases $\lambda_i^\Sigma = 0$ and $\lambda_{i,1} = 0$.
[4] See Section 1.3 for a discussion of the cases $L_i^\lambda = 0$, $P_i^\lambda = 0$, $S_i^\lambda = 0$, or $\lambda_i^\Sigma = 0$.

By segmenting the interval $[r_{\min}, r_{\max}]$ into 16 (not equally large) scales and evaluating the measure E_i^{dim} for each corresponding metric neighborhood as given in Equation (1.4), the neighborhood with smallest error $E_i^{\dim}$ over all scales is chosen. Thereby, the neighborhood is assured to favor one of the three cases: Classifying the point p_i as either a corner, an edge point, or a planar point of the geometry.

The authors then proceed to give an even more general solution for the optimal selection of neighborhood sizes. For this, recall that the eigenvalues correspond to the size of the principal components spanning a 3D covariance ellipsoid, see [PLL12]. Thus, by normalizing the eigenvalues and considering their entropy, the quantity[5]

$$E_i^{\lambda} = -\frac{\lambda_{i,1}}{\lambda_i^{\Sigma}} \ln\left(\frac{\lambda_{i,1}}{\lambda_i^{\Sigma}}\right) - \frac{\lambda_{i,2}}{\lambda_i^{\Sigma}} \ln\left(\frac{\lambda_{i,2}}{\lambda_i^{\Sigma}}\right) - \frac{\lambda_{i,3}}{\lambda_i^{\Sigma}} \ln\left(\frac{\lambda_{i,3}}{\lambda_i^{\Sigma}}\right) \tag{1.12}$$

measures the scattering of the points with respect to the covariance ellipsoid [WJM14]. Finally, as in the work of [LCL06], the measures (1.11) and (1.12) are evaluated for each point p_i with different respective neighborhood sizes r in the metric neighborhood given by Equation (1.4) and the neighborhood with the lowest value is chosen. In our evaluation of shape-aware neighborhoods, we will turn to the measures (1.11) and (1.12) in order to assess our concepts.

1.2.3 Segmentation

In the process of geometry segmentation—independent of the application setup with meshes or point sets—neighborhoods arise as regions with common properties. These segments are built to reflect the shape of the model on a coarse level. For both the mesh and the point set setting, there are thorough surveys: the authors of [Att+06] and [NL13] present the relevant developments for mesh and point set segmentation respectively. Some of the approaches presented use the concept of region growing, where after an initial selection of seed faces or points, applicants are added to a region identified by a seed point, if some conditions are fulfilled. For instance, a possible condition to rank an applicant to be added to an existing region is to determine its normal variation when compared to the normals of the points already present in the segment.

A particular segmentation problem is solved in the algorithm presented by the authors of [Hua+13]. Here, the aim is to resample a given input point set in a feature-aware manner. It is done in a three-stage process. First, reliable normals are created. Second, features are extracted and the point set is resampled away from the features following the approach of [Lip+07]. This second step yields a segmentation of the underlying geometry in piecewise flat parts. Third and finally, the features are approximated by introducing new points to sample them.

A similar idea—to extract large flat segments from the input point set—is presented in the more recent work [LB16]. The authors also deal with feature extraction, extending the idea from [CAD04], going from meshes to point sets. They use the concept of regions which respect the shape of the geometry characterized by its normals, to extract feature lines. In contrast to [Lip+07], they work with normal directions obtained from the covariance matrix (1.9). Therefore, their approach does not provide robustness against noise. Refer to Section 5.1.1 for further related work and a discussion of segmentation in the context of Variational Shape Approximation.

In consequence, some geometry segmentation algorithms utilize the concept of shape aware neighborhoods as natural ingredients. However, as stated above, all of these approaches aim at a coarse representation of the geometry. Thus, they have high tolerances in the normal variation of neighboring entities. Thereby, they create a faithful simplification of the underlying geometry.

[5] See Section 1.3 for a discussion of the cases $\lambda_{i,1} = 0$, $\lambda_{i,2} = 0$, $\lambda_{i,3} = 0$, or $\lambda_i^{\Sigma} = 0$.

Our approach does not only work on a smaller scale—the local point neighborhoods—but thereby also serves as input for any algorithm depending on local neighborhoods, as shown for point set denoising and MLS approximation in Sections 1.5.1 and 1.5.2 respectively.

1.3 Method

The works presented in the above Sections 1.2.1 and 1.2.2 all aim at improving the neighborhood information within a point set. However, they all take a similar approach, namely altering the size of the neighborhood, either heuristically or by utilizing an error functional.

We follow an idea from [Yad+18b], Section 8, to choose the size of the neighborhood not solely depending on a combinatorial value k or a metric value ε, but we build the neighborhood being aware of the geometrical shape. We assume that for each point $p_i \in P$ of the point set, we are given a normal $n_i \in \mathbb{R}^3$ and we assume them to be normalized. We further assume that the normal field on the point set consists of oriented outward normals.

Similar to [Lip+07], we face the problem of computing reliable normals without having neighborhoods at hand. Following the presented solution, we first compute normals from combinatorial or metric neighborhoods using one of the available algorithms (e.g. [MNG03; MWP18]). These initial normals are then oriented (e.g. [Hop+92]) and in the presence of noise, additional denoising is performed on the normal field (e.g. [JDZ04; Avr+10]). Thereby, we obtain the desired normal field to work with in the following.

To select neighbors of a point in a shape-aware manner we define the following sigmoid function. It is related to the sigmoid used in [Mar+06], but we fix the image of the function to be $[0,1]$ on the interval $[0,1]$.

Definition 1. *Let $a \in [0,1) \subset \mathbb{R}$, $b \in \mathbb{R}_{\geq 1} \cup \{\infty\}$, $c = b(1-a)^{-1}\pi$, and $a' = (1-a)b^{-1} + a$, i.e. $a' \in (a,1]$. Then we define the sigmoid function $\operatorname{sig}^{\cos}_{a,b}(x)$ as*

$$\operatorname{sig}^{\cos}_{a,b}(x) : [0,1] \to [0,1]$$
$$x \mapsto \begin{cases} 0 & x \in [0,a) \\ -\frac{1}{2}\cos\left(c(x-a)\right) + \frac{1}{2} & x \in [a,a') \\ 1 & x \in [a',1] \end{cases}.$$

Observe that the parameter a in Definition 1 translates the sigmoid along the x-axis and b scales the incline of the cosine-curve. Thus, the sigmoid can be used to model a sharp cutoff at an arbitrary point $a \in [0,1)$ by using $b = \infty$. Utilizing this sigmoid, we obtain the weights

$$w^{\operatorname{sig}}_{ij}(a,b) = \operatorname{sig}^{\cos}_{a,b}\left(\frac{\langle n_i, n_j\rangle + 1}{2}\right), \tag{1.13}$$

for given values a and b. Recall that we asked for normalized outward normals. Thereby, very similar normals enter as argument close to 1, while opposing normals have a scalar product of -1 s.t. the argument of $\operatorname{sig}^{\cos}_{a,b}$ is 0. In combination with the neighborhoods (1.1) or (1.4), these weights create a geometrically weighted covariance matrix (see [HBC11]) by

$$C^{\operatorname{sig}}_i(a,b) = \sum_{p_j \in \mathcal{N}_k(p_i)} w^{\operatorname{sig}}_{ij}(a,b)(p_j - \bar{p}_i)(p_j - \bar{p}_i)^T, \tag{1.14}$$

where again $\bar{p}_i = \frac{1}{k}\sum_{p_j \in \mathcal{N}_k(p_i)} p_j$. Observe that in dependence on the parameters a and b the covariance matrix might degenerate when all weights are equal to 0. In other words, choosing a too

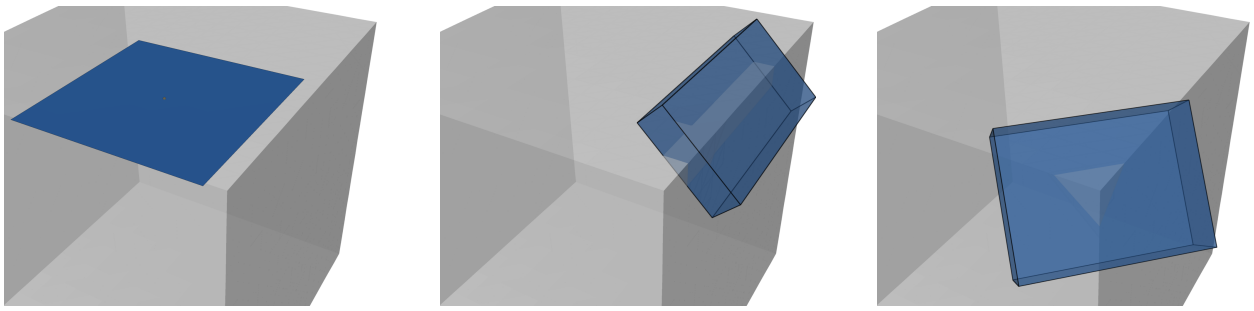


Figure 1.2: *Illustration of the behavior of the eigenvalues obtained from the PCA at the different locations, i.e. a planar region, edge, and corner. All of these draw a box around the marked vertex of the cube model with 1906 vertices, with its expansion linked to the magnitude of the three eigenvalues. In the image, we set* $k = 20$ *for* $\mathcal{N}_k(p_i)$.

high value for a and thereby shifting the curve too far towards 1 reduces the number of neighbors receiving a strictly positive weight w.r.t. the normal similarity. Therefore, the parameter a has to be chosen carefully.

Given weights (1.13) such that the weighted covariance matrix (1.14) does not degenerate, note that $C_i^{\text{sig}}(a, b)$ is symmetric and positive-semi-definite. This is because the weights are symmetric, i.e. $w_{ij}^{\text{sig}}(a, b) = w_{ji}^{\text{sig}}(a, b)$. Therefore, the weighted covariance matrix gives rise to eigenvalues $\tilde{\lambda}_{i,1} \geq \tilde{\lambda}_{i,2} \geq \tilde{\lambda}_{i,3} \geq 0$. These in turn can be used in the evaluation of measures (1.11) and (1.12). However, this is not unconditionally possible. In particular on CAD models, cases $\tilde{\lambda}_{i,1} = \tilde{\lambda}_{i,2}$, $\tilde{\lambda}_{i,2} = \tilde{\lambda}_{i,3}$, or $\tilde{\lambda}_{i,3} = 0$ can arise, see Figure 1.2. Depending on the occurrence of any of these cases, we set the related term in Equation (1.11) to 0. We proceed similarly in the case of measure (1.12): The sum of the eigenvalues cannot be 0, because we assumed a non-degenerated covariance matrix and thereby we have at least one strictly positive eigenvalue. But it might occur that $\tilde{\lambda}_{i,3} = 0$ or even $\tilde{\lambda}_{i,2} = 0$. The latter happens, when we collect neighbors along a straight line in $\mathbb{R}^3$. If any of these problematic cases occur, we once more neglect the related terms in Equation (1.12) and calculate only the remaining ones.

Note that all considerations are due to the fact, that equal eigenvalues arise from the identical propagation of the neighborhoods in multiple directions and eigenvalues with magnitude zero indicate no propagation in the related direction at all. Neglecting the relevant terms in both measures (1.11) and (1.12) is owed to the fact that the natural logarithm is not defined at zero. In the original publication [WJM14], these modifications were probably unnecessary due to the noise levels in the considered geometries.

Using weights (1.13) with a high parameter b for the sigmoid will result in a sharp cutoff. Thus, points will not only be assigned a low, but a complete 0 weight. When disregarding these points, connectedness in a neighborhood weighted by (1.13) no longer means necessarily geometrical closeness. See Figure 1.3 for an example, in which the weighted neighborhood collects neighbors being similar w.r.t. point normals and the result is visually disconnected.

1.4 Experimental Results

We evaluate our neighborhood concept on several synthetic (fandisk, bearing) and real-world models (bunny, kitten). Compared to [WJM14], we use the error measures (1.11) and (1.12), with the modifications mentioned in Section 1.3. The experiments evaluate the behavior of several sigmoids generated by different parameter choices for a and b as well as results deduced by equal weights all set to one ($a = 0$, $b = \infty$) and a sharp cut-off as used in [Yad+18b], ($a = \rho$, $b = \infty$),

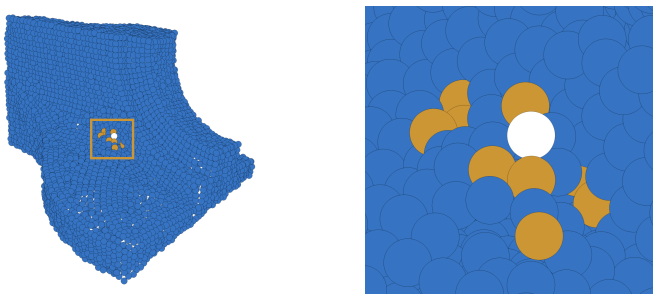


Figure 1.3: *A point on the fandisk model and its cut-off neighborhood for a large parameter b. Note how the visual impression of connectivity is lost and the neighborhood seems to consist of several separate parts.*

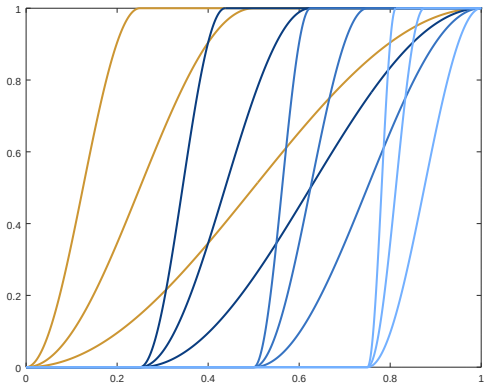


Figure 1.4: *Sigmoids for different values of a and b, see Definition 1.*

with threshold ρ from [Yad+18b], see Section 8. Figure 1.4 illustrates different sigmoids for $\{(a,b) \mid a \in \{0, 0.25, 0.5, 0.75\}, b \in \{1,2,4\}\}$ in yellow ($a = 0$), dark blue ($a = 0.25$), blue ($a = 0.5$), and light blue ($a = 0.75$) respectively. The extreme cases where $b = \infty$ are not shown, as they simply "jump" from 0 to 1 at a.

We run the experiments as follows: For each $k \in \{6, \dots, 20\}$ and the resulting neighborhood (1.1) as well as for each of the sigmoids given above and an additional set of parameters with $a = 0.9$ (default parameter for ρ in [Yad+18b], Section 8), we compute the measures (1.11) and (1.12) at every point p_i of the given point set. Then, for each choice of parameters (a,b), we pick two neighborhood sizes $k_i^{\dim}, k_i^{\lambda}$ such that the error measures are minimal respectively at the point p_i:

$$k_i^{\dim} = \arg\min_{k\in\{6,\dots,20\}} E_i^{\dim},$$
$$k_i^{\lambda} = \arg\min_{k\in\{6,\dots,20\}} E_i^{\lambda}.$$

This yields a local neighborhood size at each point p_i for a given sigmoid obtained from (a,b). Then, for each parameter choice (a,b), we consider the average measure over all points of the point set:

$$E_{avg}^{\dim} = \frac{1}{n}\sum_{i=1}^{n}\left(\min_{k\in\{6,\dots,20\}} E_i^{\dim}\right), \tag{1.15}$$

$$E_{avg}^{\lambda} = \frac{1}{n}\sum_{i=1}^{n}\left(\min_{k\in\{6,\dots,20\}} E_i^{\lambda}\right). \tag{1.16}$$

Aside from the average, we also consider the minimum, maximum, and standard deviation which are given for all models and both error measures in Appendix B.

1.4.1 CAD Models

At first, we are going to consider the noiseless CAD models bearing and fandisk, see Figures B.1 and B.2. Both models carry a variety of features, such as curved edges, corners, creases, etc.

Consider Tables B.1 and B.2 for a comparison of the obtained measures for different choices of parameters a and b. Both CAD models obtain the smallest measures when evaluated with a relatively high comparison parameter ($a \geq 0.75$) and a soft increase ($b = 1$). This contrasts the use of a sharp cutoff as utilized in [Yad+18b], Section 8. See Section 1.5.1 for a more detailed evaluation in the context of this application. Observe that both models obtain lowest measures for the highest a possible with soft increase, as long as the collected neighborhoods are not empty. In Table B.1, we can see that $a = 0.9$ is too strict for the given model, such that in consequence for at least one point p_i of the model all neighborhoods for $k \in \{6, \dots, 20\}$ are empty. In this case, we do not report the results as the computation clearly failed.

1.4.2 Real-World Models

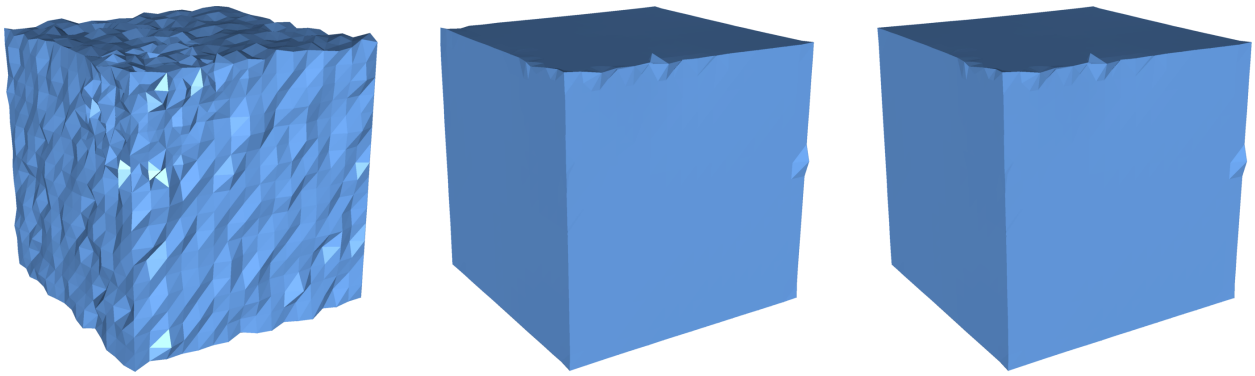
Additionally to the artificial CAD models, we run our experiments on real-world models arising from 3D scanners which introduce noise during the acquisition process. We use the bunny (Figure B.3) and kitten model (Figure B.4).

Here, we find a similar behavior. For both real-world models, their results are given in Tables B.3 and B.4. Once more, as high as possible values of the comparison parameter $a \in \{0.75, 0.9\}$ yield the lowest measures, while also a soft increase is favored ($b \in \{1, 2\}$).

1.4.3 Assessment of Results

Our numerical experiments show that all considered models, being them CAD or real-word, ask for the highest comparison parameter a possible and soft increases b of the sigmoid curve. Note that the authors of [WJM14] also use that neighborhood size k which gives the lowest value of $E^{\dim}$ and E^{λ} respectively. However, they do not incorporate weights, that is they assign each possible neighbor a weight of 1. This is exactly achieved in our setup for $a = 0$ and $b = \infty$. Hence, in Tables B.1–B.4, the measures attained by the approach of [WJM14] are given in the rightmost cell of the first row. Note that none of these attain smallest measure $E^{\dim}$ or E^{λ} in any of our experiments.

Furthermore, in [Yad+18b], Section 8, we favor a sharp cut-off. In the setup presented here, this corresponds to parameters $a = 0.9$ and $b = \infty$. For two of the four geometries considered, these parameters yield empty neighborhoods for at least one point of the geometry. In the remaining two cases, only for the bunny model, the least standard deviation is attained for the sharp cut-off in the measure $E^{\dim}$, see Table B.3. In all other cases, our sigmoid approaches attain smaller measures. In the following, we will evaluate our shape-aware neighborhood concepts within two applications to validate the experimental results.

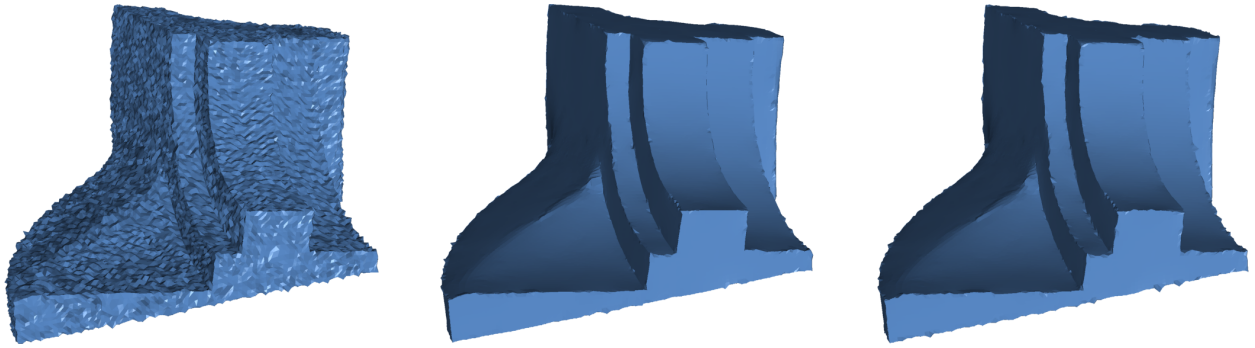


(a) *Noisy cube*

(b) *Denoised with sharp cut-off.*

(c) *Denoised with sigmoid soft increase.*

Figure 1.5: *Visual comparison of denoising following [Yad+18b], Section 8, with original sharp cut-off and a soft increasing sigmoid.*



(a) *Noisy fandisk*

(b) *Denoised with sharp cut-off.*

(c) *Denoised with sigmoid soft increase.*

Figure 1.6: *Visual comparison of denoising following [Yad+18b], Section 8, with original sharp cut-off and a soft increasing sigmoid.*

1.5 Applications

1.5.1 Application: Point Set Denoising

In [Yad+18b], Section 8, we propose a three-staged algorithm for point set denoising, where we use a sharp cut-off to detect neighborhoods. Here, similar point normals receive weight 1 w.r.t. a user-given threshold ρ, while non-similar normals are assigned weight 0. The leftmost images of Figures 1.5 and 1.6 show the cube and fandisk models equipped with Gaussian noise $\sigma \approx 0.2\ell_{nn}$ and $\sigma \approx 0.33\ell_{nn}$, respectively. The value ℓ_{nn} describes the average distance of all one-nearest-neighbor distances of all points in the point set. The second image in both figures gives the results using the denoising algorithm described in [Yad+18b], Section 8, with parameters $\rho = 0.95$, $I = 150$ iterations (cube) and $\rho = 0.9$, $I = 50$ iterations (fandisk), while the remaining parameters are kept default. The third image in both figures shows the results, when we replace the sharp cut-off with our sigmoid setting $a = 0.95$ (cube), $a = 0.9$ (fandisk) and $b = 1$ for both models, which results in a soft increase instead of a sharp cut-off.

A comparison of the results gained with the cut-off used in [Yad+18b], Section 8, and a soft increase with our sigmoid is displayed in Table 1.1. Here, we use the *Metro* algorithm of [CRS98] available as part of [Cig+08]. We present the values: minimum distance (Min), maximum dis-

Model	Nhd	Min	Max	Mean	RMS
Cube	Cut-off	0	0.011415	0.000775	0.001035
	Sigmoid	1.E-6	0.005939	0.000746	0.000968
fandisk	Cut-off	1.E-6	0.018277	0.004145	0.006250
	Sigmoid	1.E-6	0.017518	0.004264	0.006085

Table 1.1: *Results of the Hausdorff distances (given as Min, Max, Mean, RMS) measured between denoised cube and fandisk models (using sharp cut-off and softer increase) and their clean representatives using the* Metro *algorithm of [CRS98] available as part of [Cig+08].*

tance (Max), mean distance (Mean), and root mean square distance (RMS). Note that the values are taken w.r.t. the bounding box diagonal.

Both models and the chosen parameters are proven to yield very good results in [Yad+18b], Section 8. When we replace the binary weight assignment in the neighborhood detection in Stage 1 of the iterative procedure and do not jump to 1 at ρ, but let the curve increase softly, this gives comparable, or even better results, see Table 1.1.

From this exploration, we see that the general concept of shape-aware neighborhoods, independent of the specific parameters used, already provides good results. This was shown in an extensive comparison to other state-of-the-art methods in [Yad+18b]. As the denoising method of [Yad+18b], Section 8, already incorporates shape-aware neighborhoods, the following question remains: How much does a non-shape-aware application gain from the presented concept? In order to answer this question, we turn to the MLS framework.

1.5.2 Application: Moving Least Squares (MLS)

We will now evaluate the different neighborhood concepts in the context of a non-shape-aware application, namely the MLS framework ([Lev98; Lev04; Ale+01; SL16]). The MLS procedure takes a point set as input and locally computes a best-fitting smooth manifold approximating the given point set. Naturally, the obtained result does not reflect features of the input geometry. However, in many applications, it is important to have a smooth representation everywhere but at the features, which can for example be obtained by the *Robust Implicit Moving Least Sqaures* (RIMLS) approach of [ÖGG09]. We will incorporate the concept of shape-aware neighborhoods to the MLS approach and then compare the results to RIMLS.

For implementing the MLS, we follow the description of [SL16, Sections 3.1 and 3.2]. The procedure outlined there to perform the MLS projection for a point p_i consists of two steps: Finding the local coordinates and performing the MLS projection. Both include the use of a non-negative rapidly decreasing weight function $\theta(\|p_j - q\|)$, where q is the sought for projection of p_i and p_j are its neighbors. In order to incorporate our sigmoid weights (1.13), we replace each occurrence of θ by $\theta \cdot w_{ij}^{\text{sig}}(a, b)$. Furthermore, we incorporate them in the evaluation of the polynomial approximation in step 2. For a more thorough introduction to MLS, see Section 4.4.1.

The effect of the incorporation of our weights into the MLS procedure can be seen in Figure 1.7. The input geometry consists of two evenly sampled planes with normals pointing away from the respective other. Consider now a point p_i on one of the two planes. When using the standard k nearest neighborhoods (1.1), the neighborhood of p_i will include points from both planes, see Figure 1.7a. Thus, when performing the MLS procedure, the two planes are merged into one, as can be seen in Figure 1.7a. In contrast, when using shape-aware neighborhoods, the neighbors of p_i all live on the same plane. Thus, after performing the MLS procedure, the two planes stay separated, see Figure 1.7b.

(a) *k nearest neighbors (top), MLS projection (bottom)*

(b) *shape-aware neighbors (top), MLS projection (bottom)*

Figure 1.7: *MLS procedure using k nearest neighbors (left) and shape-aware neighborhoods (right). The upper row shows the neighborhood for the marked dark blue point, and the lower row shows the result after applying MLS to all points of the two planes. Note how the neighborhood includes points from both planes when using k nearest neighbors and only points from the plane of the query point when using shape-aware weights.*

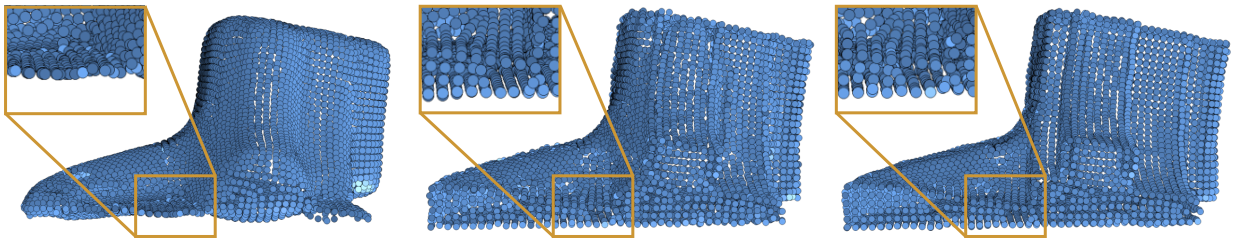


Figure 1.8: *Comparison of MLS (left), MLS with shape-aware neighborhoods (center), and RIMLS of [ÖGG09] (right).*

For a more practical example, we apply the MLS procedure to they fandisk model corrupted by Gaussian noise ($\sigma \approx 0.26\ell_{nn}$, with ℓ_{nn} being the average one-nearest-neighbor distance). The result is seen in the left image of Figure 1.8. As highlighted, the narrow part of the model is glued together by the MLS projection. The center image shows the modified MLS procedure incorporating our shape-aware weights. Note how the narrow segment keeps separated. For comparison, the right image shows the application of RIMLS.

A comparison of the results gained with the RIMLS approach and the modified MLS with our sigmoid is displayed in Table 1.2. Again, we use the *Metro* algorithm of [CRS98] available as part of [Cig+08]. We present the values: minimum distance (Min), maximum distance (Max), the mean distance (Mean), and the root mean square distance (RMS). Note that the values are taken w.r.t. the bounding box diagonal.

From the results of Table 1.2, we see that RIMLS outperforms our simple approach of solely including shape-aware neighborhoods into the MLS procedure. Visually, this effect becomes apparent in Figure 1.8, where the right image obtained by RIMLS is smoother than the middle image from MLS with shape-aware neighborhoods. Nonetheless, our approach is easily included in the MLS pipeline and does not differ from RIMLS by an order of magnitude. Therefore, further research has to be invested in tuning this procedure in order to compete with RIMLS and similar other approaches.

Model	Nhd	Min	Max	Mean	RMS
cube (clean)	RIMLS	0	0.014427	0.002916	0.003696
	Sigmoid	0	0.018776	0.002618	0.003622
cube (noisy)	RIMLS	0	0.014427	0.002890	0.003704
	Sigmoid	0	0.029943	0.003352	0.004447
bearing	RIMLS	0	0.023033	0.002109	0.002768
	Sigmoid	0	0.043271	0.003344	0.005272
fandisk	RIMLS	0	0.007133	0.001227	0.001541
	Sigmoid	0	0.024058	0.002246	0.003433

Table 1.2: *Results of the Hausdorff distances (given as Min, Max, Mean, and RMS) measured between the results of RIMLS, see [ÖGG09], and MLS utilizing shape-aware neighborhoods with the clean model respectively.*

1.6 Conclusion

In this section, we followed up on an idea of [Yad+18b], see Section 8, and formulated the concept of neighborhoods that do not aim for a heuristically chosen size or try to obtain an optimal size by a specialized error measure. Our neighborhoods rather adapt to the shape of the geometry. The presented approach was evaluated experimentally with measures $E^{\dim}$ and E^{λ} derived in the work of [WJM14]. In our experiments on both CAD and real-world models, we found that incorporating sigmoid weights into the computation yields smaller error measures than the uniform weights from [WJM14] or the sharp cut-off neighborhoods used in [Yad+18b], Section 8.

We further evaluate our approach in two application contexts. First, we implement it in the denoising setup of [Yad+18b]. The changes in results are minor here, as the algorithm already utilizes shape-aware neighborhoods, only with a sharp cut-off. Therefore, we secondly embed our concept in the MLS procedure of [SL16]. Here, we show that undesired effect of MLS such as the merging of two originally separated parts of a geometry can efficiently be prohibited. Furthermore, we compare to the RIMLS approach of [ÖGG09] and show that our method can compete with it visually and quantitatively, even though it is derived only from first principles and does not utilize an involved statistical framework like RIMLS does. However, RIMLS still provides smoother results. Thus, further research has to be invested in fine-tuning MLS with shape-aware neighborhoods.

2 k-d Trees

In the previous section, we have presented several notions of neighborhoods for point sets. For practical applications, these concepts are not of any use if they cannot be computed efficiently. Thus, in this section, we give an introduction to the data structure of k-d trees, first presented by Friedman, Bentley, and Finkel in 1977, see [FBF77]. After a short introduction to the data structure (Section 2.1), we turn to the proof of efficiency by Friedman and his colleagues (Section 2.2). The main contribution of this section is:

- Translation into modern terms and elaboration of the proof of Friedman, Bentley, and Finkel (Section 2.2).

2.1 The Data Structure of k-d Trees

In this section, we will give a brief introduction to the k-d tree data structure. It was originally presented by Jon Louis Bentley in 1975, see [Ben75]. A modern introduction to the two-dimensional case can be found in [Ber+00, Chapter 5.2].

Let $P = \{p_1, \dots, p_n\} \subset \mathbb{R}^d$ be a finite point set with $p_i = (p_i^1, \dots, p_i^d)^T \in \mathbb{R}^d$ for $i = 1, \dots, n$. A k-d tree for P is defined recursively. If P is empty or contains only one point, an empty tree or a tree with one node containing the one point is returned. Otherwise, it is determined in which dimension $d' \in [d]$ the point set has the largest spread. That is, d' is chosen such that there are two points p_i, p_j with $\left|p_i^{d'} - p_j^{d'}\right| \geq \left|p_\ell^{d''} - q_m^{d''}\right|$ for all $d'' \in [d]$, $\ell, m \in [n]$. Now, find the median[6] of the points p_i according to a sorting along this dimension, i.e. $p_{i_1}^{d'} \leq \ldots \leq p_{i_{\lceil n/2 \rceil}}^{d'} \leq \ldots \leq p_{i_n}^{d'}$, denote it by $q = p_{i_{\lceil n/2 \rceil}}$. Note that the points do not necessarily have to be sorted, as the median can be found in linear time, see [Blu+73]. However, there has to be a unique ordering on the points to determine the resulting k-d tree uniquely. Thus, we assume that the point set P can be uniquely ordered[7] along any dimension $d' \in [d]$. After finding the median, a hyperplane $H = \{x \in \mathbb{R}^d \mid x^{d'} = q^{d'}\}$ is introduced, which splits the set P into two subsets

$$P_1 = \{p_{i_1}, \dots, p_{i_{\lceil n/2 \rceil - 1}}\}, \qquad P_2 = \{p_{i_{\lceil n/2 \rceil + 1}}, \dots, p_{i_n}\}$$

with P_1 containing at most one point more than P_2. A node is created, holding q and H. Partitioning P into the two subsets P_1 and P_2 can be performed in $\Theta(n)$. The node is given the results of recursively processing P_1 and P_2 as children and then it is returned. An example for the building process in the two-dimensional case is given in Figure 2.1. The algorithm is given in pseudo code as Algorithm 1. From this building procedure, we see that the building time $T(n)$ of a k-d tree satisfies the following recursion

$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ \Theta(n) + 2 \cdot T(\lceil n/2 \rceil) & n > 1 \end{cases},$$

which solves to $T(n) = \Theta(n \cdot \log(n))$, see [SW11, pp. 272–274]. By the above and by noting that each node of the k-d tree stores a distinct point of the input set P, we proved the following theorem.

[6] For any number $n \in \mathbb{N}$ of ordered points, we define the median to be the point with index $\lceil n/2 \rceil$. Note that $\lfloor n/2 \rfloor + 1$ would also give a valid choice which has to be considered slightly differently when building the k-d tree.

[7] In any practical application, this can be achieve by e.g. sorting points with equal entries according to their indices, i.e. in the case $p_i = p_j$ we order by $i < j$.

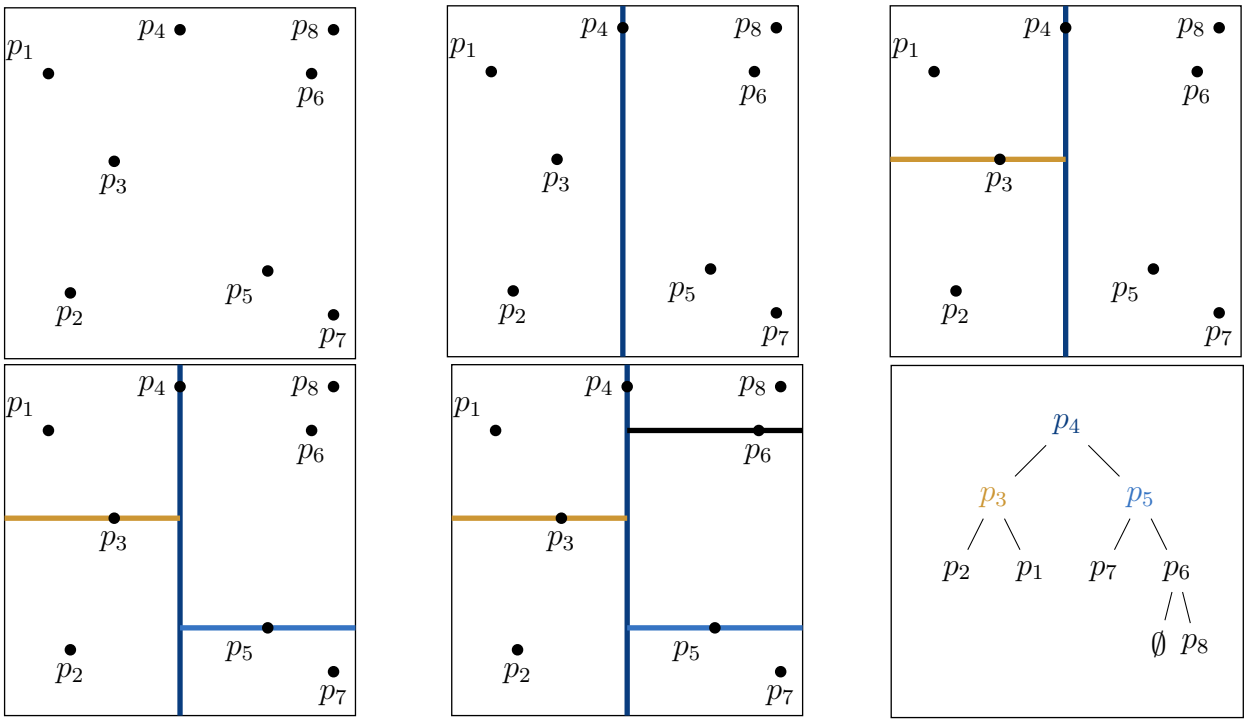


Figure 2.1: *Recursively building a k-d tree on eight points. The hyperplanes are shown in the first five figures, while the whole tree is shown in the last figure on the lower right.*

Theorem 1 (Storage requirement and building time of a k-d tree, [Ber+00]). *A k-d tree for a set of n points uses $\Theta(n)$ storage and can be constructed in $\Theta(n \cdot \log(n))$ time.*

The k-d tree data structure has applications in orthogonal range searches, as discussed in [Ber+00, Chapter 5.2] and [Skr14b, Chapter 3.2]. In the following, we will focus on applications in the context of neighborhood queries.

Algorithm 1 Build k-d tree

procedure BUILD K-D TREE(point set P)
  if $|P| \leq 1$ **then**
    return node containing P
  end if
  $d' \leftarrow$ most spread dimension of P
  $q \leftarrow$ median according to dimension d'
  $P_1 \leftarrow \{p_i \in P \mid p_i^{d'} \leq q^{d'}, p_i \neq q\}$
  $P_2 \leftarrow P \backslash (P_1 \cup \{q\})$
  $H \leftarrow \{x \in \mathbb{R}^d \mid x^{d'} = q^{d'}\}$
  $N_\ell \leftarrow$BUILD K-D TREE(P_1)
  $N_r \leftarrow$BUILD K-D TREE(P_2)
  return node containing q, H with N_ℓ and N_r as children
end procedure

2.2 Neighborhood Queries in Logarithmic Time

Amongst the data structures computing the k nearest neighborhood (1.1), the most prominent choice is the k-d tree data structure presented above. The reason is that in 1977, Friedmann, Bentley, and Finkel were able to prove an average case running time of $\mathcal{O}(\log(n))$ for a single neighbor query in a k-d tree built on n points. The proof in their paper is very concise and covers roughly three pages [FBF77, pp. 214–216]. Thus, we will give a more elaborate version of their proof in modern wording here.

Let a point set P and a corresponding k-d tree built according to Section 2.1 be given. A neighborhood query for any point $p \in \mathbb{R}^d$ is then performed by traversing the tree to the leaf representing the box which contains the query point. From there, the query goes back to the root, investigating subtrees along the path where the splitting hyperplane is closer to p than the currently found nearest neighbors. When reaching a node, all elements in it are investigated as to whether they are closer to p than the current closest points found. The algorithm is fast, as it can be expected that several subtrees do not have to be investigated. See Algorithm 2 for a pseudo code version and see [Skr14b, Section 5.2] for an illustration of the nearest neighbor search for a single nearest neighbor, i.e. $k = 1$.

We will now investigate the following question: What is the expected query time for a neighborhood query in a k-d tree? We will do this by considering a different setup compared to the construction in Section 2.1. Namely, we will store points only in the leaves of the tree and we will further allow for more than one point to be stored in each leaf. Collecting all factors that can affect the runtime of the neighborhood search, we get the following list:

- total number n of points,
- dimension d of the ambient space,
- number of neighbors sought k,
- number b of points to be stored in each leaf,
- distance measure[8] $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_{\geq 0}$, $(p, q) \mapsto d(p, q)$,
- density $\delta : \mathbb{R}^d \to [0, 1]$ of points in space.

The density is taken into account as a way to analyze an arbitrary point set. As we will not make any assumption on the actual positions of the points $p_i \in P$ throughout the proof, we turn to a probabilistic argument. Thus, we consider an arbitrary non-empty sample space Ω and random variables $X_1, \ldots, X_n, X_p : \Omega \to \mathbb{R}^d$. Draw a sample $\omega \in \Omega$. Now, our point set P is given by $P = \{X_1(\omega), \ldots, X_n(\omega)\}$ and the point p to search neighbors for is given as $X_p(\omega)$. That is, we want to find k points from $\{X_1(\omega), \ldots, X_n(\omega)\} \subset \mathbb{R}^d$ that are the k nearest neighbors to $X_p(\omega) \in \mathbb{R}^d$ within the set $\{X_1(\omega), \ldots, X_n(\omega)\} \subset \mathbb{R}^d$.

Without loss of generality we assume $X_1(\omega), \ldots, X_k(\omega)$ to be the k nearest neighbors with $X_k(\omega)$ being the farthest from $X_p(\omega)$. Now, denote

$$B_k(X_p(\omega)) = \{x \in \mathbb{R}^d \mid d(x, X_p(\omega)) \leq d(X_k(\omega), X_p(\omega))\}$$

to be the ball around $X_p(\omega)$ containing all points in $\mathbb{R}^d$ with distance less than or equal to the distance to $X_k(\omega)$. The volume of this ball is

$$v_k(X_p(\omega)) := \int_{B_k(X_p(\omega))} 1 \, dx.$$

[8] See Footnote 1, page 8.

Algorithm 2 Nearest Neighbor Search in k-d trees

```
1: procedure NNk-dTree(point p, k-d tree T, distance ε, number k)
2:     L ←empty list
3:     return NNk-dTreeRec(p,root,ε, k, L)
4: end procedure

5: procedure NNk-dTreeRec(point p, k-d tree T, distance ε, number k, list L)
6:     if T = ∅ then
7:         return L
8:     end if
9:     Extract point p_j from T.root and store it in L if ||p_j − p|| ≤ ε.
10:    if L is larger than k then
11:        Delete the point with largest distance to p from L.
12:    end if
13:    if T is just a leaf then
14:        return L
15:    end if
16:    if T.root.leftSubtree contains p then
17:        T_1 = T.root.leftSubtree, T_2 = T.root.rightSubtree
18:    else
19:        T_2 = T.root.leftSubtree, T_1 = T.root.rightSubtree
20:    end if
21:    NNk-dTreeRec(p, T_1, ε, k, L)
22:    if |L| < k and ||p − T.root.hyperplane|| < ε then
23:        NNk-dTreeRec(p, T_2, ε, k, L)
24:    else if ||L.farthest − p|| > ||p − T.root.hyperplane|| and ||p − T.root.hyperplane|| ≤ ε
   then
25:        NNk-dTreeRec(p, T_2, ε, k, L)
26:    end if
27:    return L
28: end procedure
```

Furthermore, the probability content of this region—according to the density $\delta(x)$—is

$$u_k(X_p(\omega)) = \int_{B_k(X_p(\omega))} \delta(x)\, dx.$$

Since $\int_{\mathbb{R}^d} \delta(x)dx = 1$, we have $u_k(X_p(\Omega)) \leq 1$. Furthermore, since $\delta(x) \geq 0$ for all $x \in B_k(X_p(\omega))$, we have $0 \leq u_k(X_p(\omega))$. Consider some additional random variable $X : \Omega \to \mathbb{R}^d$ with density δ. Then,

$$\begin{aligned} u_k(X_p(\omega)) &= \int_{B_k(X_p(\omega))} \delta(x)\, dx \\ &= \int_{\mathbb{R}^d} \delta(x) \cdot \mathbb{1}_{B_k(X_p(\omega))}(x)\, dx \\ &\overset{\diamond}{=} \mathbb{E}(\mathbb{1}_{B_k(X_p(\omega))}(X)), \end{aligned}$$

where $\diamond$ holds by the law of the unconscious statistician: $\mathbb{E}(g(X)) = \int_{\mathbb{R}^d} g(x) \cdot \delta(x)\, dx$, [BH14, p. 156], with $g(x) := \mathbb{1}_{B_k(X_p(\omega))}(x)$. Furthermore, we have

$$\mathbb{1}_{B_k(X_p(\omega))}(x) = \begin{cases} 1, & \text{if } x \in B_k(X_p(\omega)) \\ 0, & \text{otherwise} \end{cases}.$$

Therefore,

$$\begin{aligned} \mathbb{E}(\mathbb{1}_{B_k(X_p(\omega))}(X)) &= \mathbb{P}(X \in B_k(X_p(\omega))) \\ &= \mathbb{P}(d(X, X_p(\omega)) \leq d(X_k(\omega), X_p(\omega))). \end{aligned}$$

Now, from X and X_i we define new random variables

$$\xi := d(X, X_p(\omega)), \qquad \xi_i := d(X_i, X_p(\omega)),$$

which provide distances

$$\xi_i(\omega) := d(X_i(\omega), X_p(\omega)).$$

Then, $u_k(X_p(\omega)) = \mathbb{P}(\xi \leq \xi_k(\omega))$ by the above computations. Furthermore, $\xi : \Omega \to \mathbb{R}$ is a random variable with distribution $F_\xi(x) = \mathbb{P}(\xi \leq x)$, therefore $u_k(X_p(\omega)) = F_\xi(\xi_k(\omega))$. The question to be answered now is: What is the distribution F_ξ at $\xi_k(\omega)$?

Order the distances $\xi_i(\omega)$, then we have the following one-to-one correspondence:

$$\xi_1(\omega) \leq \ldots \leq \xi_n(\omega) \overset{1:1}{\longleftrightarrow} F_\xi(\xi_1(\omega)) \leq \ldots \leq F_\xi(\xi_n(\omega)).$$

Now, consider the random variable $F(\xi_i) : \Omega \to [0, 1]$. It is defined by

$$F(\xi_i)(\omega) := F_\xi(\xi_i(\omega))$$

and its distribution is

$$\mathbb{P}(F(\xi_i) \leq x) = \mathbb{P}(\xi_i \leq F_\xi^{-1}(x)) \overset{\star}{=} F_{\xi_i}(F_\xi^{-1}(x)) \overset{\diamond}{=} x,$$

where $\star$ holds because $F_{\xi_i}(x) = \mathbb{P}(\xi_i \leq x)$ and where $\diamond$ holds because ξ and ξ_i have the same density δ and thus the same distribution $F_\xi = F_{\xi_i}$. Hence, a single $F(\xi_i)$ is uniformly distributed.

However, when choosing an ordering, the k-th distance ξ_k is β-distributed (see Appendix C), therefore

$$u_k(X_p(\omega)) = \beta(\omega).$$

It can be shown that

$$u_k(X_p(\omega)) = \frac{k}{(n+1)}, \tag{2.1}$$

see Appendix C. This states that any compact volume enclosing exactly k points has probability content $\frac{k}{(n+1)}$ on average. Now, assume that n is large enough such that $B_k(X_p(\omega))$ is small and thus $\delta(x)$ can be approximated by a constant $\bar{p}(X_p(\omega))$ on $B_k(X_p(\omega))$. In this case

$$\int_{B_k(X_p(\omega))} \delta(x)\, dx = u_k(X_p(\omega)) \approx \bar{p}(X_p(\omega)) \cdot v_k(X_p(\omega)). \tag{2.2}$$

Assume further that $\delta(x)$ is continuous (which enables us to approximate it over a small region in the first place), then by definition of $B_k(X_p(\omega))$, there are small neighborhoods $B_{\varepsilon_i}(X_i(\omega))$, $i = 1, \ldots, k-1$ such that $\delta(x) > 0$ for all $x \in B_{\varepsilon_i}(X_i(\omega))$, $i = 1, \ldots, k-1$. Therefore,

$$\bar{p}(X_p(\omega)) = \int_{B_k(X_p(\omega))} \delta(x)\, dx \overset{\delta(x)\geq 0\ \forall x\in\mathbb{R}^d}{\geq} \sum_{i=1}^{k-1} \underbrace{\int_{B_{\varepsilon_i}(X_i(\omega))} \delta(x)\, dx}_{>0} > 0.$$

Now, we obtain

$$\frac{k}{(n+1)} \overset{(2.1)}{=} u_k(X_p(\omega)) \overset{(2.2)}{\approx} \bar{p}(X_p(\omega)) \cdot v_k(X_p(\omega)),$$

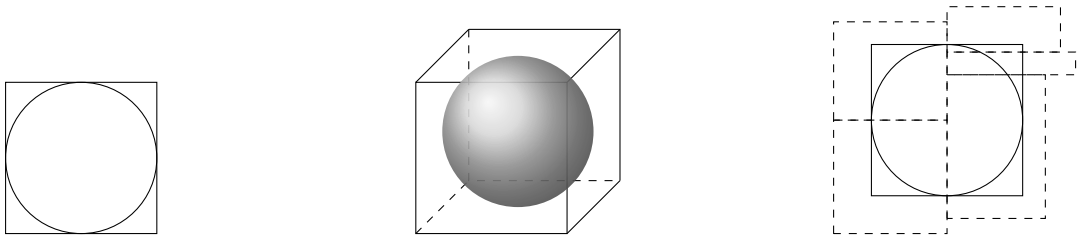
which is equivalent to

$$v_k(X_p(\omega)) \approx \frac{k}{(n+1) \cdot \bar{p}(X_p(\omega))}. \tag{2.3}$$

Observe now the effect of the k-d tree partitioning algorithm described in Section 2.1. Choosing the median ensures that the bucket sizes of all non-empty buckets will be between $\lceil b/2 \rceil$ and b, where b is the maximum bucket size. Choosing to split on the widest spread dimension ensures that the geometric shape of these buckets will be reasonably compact. In fact, the expected edge lengths of these buckets at most differ pairwise by a factor of 2. The buckets themselves are by this approximation hypercubical with edge length equal to the d-th root of the volume of space occupied by the bucket. The edges are parallel to the coordinate axes. The effect of the k-d tree partitioning, then, is to divide the coordinate space into approximately hypercubical subregions, each containing the aforementioned roughly same number of records. Because of (2.3) we have that the expected volume of such a bucket is

$$\mathbb{E}(v_b(X_b)) \approx \frac{b}{(n+1) \cdot \bar{p}(X_b)}, \tag{2.4}$$

where $X_b : \Omega \to \mathbb{R}^d$ is a random variable yielding a point that locates the bucket in the coordinate space.



(a) *Ball B_k and Cube C_k for $d = 2$.*

(b) *Ball B_k and Cube C_k for $d = 3$.*

(c) *A hypercube C_k overlapped by $\bar{\ell} = 5$ buckets.*

Figure 2.2: *Illustration of hypercubes in different dimensions.*

Consider now the smallest d-dimensional hypercube $C_k(X_p(\omega))$ with edges parallel to the coordinate axes that completely contains the ball $B_k(X_p(\omega))$. The volume $V_k(X_p(\omega))$ of this hypercube is proportional to $v_k(X_p(\omega))$, with proportionality constant

$$\frac{\text{volume}(d-\text{dim cube})}{\text{volume}(d-\text{dim ball})} = \frac{(2r)^d}{r^d \cdot \frac{\pi^{d/2}}{\Gamma(d/2+1)}} = \frac{2^d \cdot \Gamma(d/2+1)}{\pi^{d/2}} =: G(d),$$

where r is the radius of the ball. See Figures 2.2a and 2.2b for an illustration of B_k and C_k. Therefore,

$$V_k(X_p(\omega)) = G(d) \cdot v_k(X_p(\omega)) \overset{(2.3)}{\approx} \frac{G(d) \cdot k}{(n+1) \cdot \bar{p}(X_p(\omega))}. \tag{2.5}$$

In order to calculate the average number of buckets $\bar{\ell}$ examined by the k-d tree during a search for the k nearest neighbors, we need to find the buckets overlapping the ball $B_k(X_p(\omega))$, see Figure 2.2c for an illustration. This number $\bar{\ell}$ is bounded from above by $\bar{L}$, the number of buckets overlapping the hypercube $C_k(X_p(\omega))$, where

$$\bar{L} = \begin{cases} \left(\left\lfloor \frac{e_k(X_p(\omega))}{e_b(X_p(\omega))} \right\rfloor + 1 \right)^d & \text{for } e_b(X_p(\omega)) \leq e_k(X_p(\omega)) \\ 2^d & \text{otherwise} \end{cases}$$

where $e_k(X_p(\omega))$ denotes the edge length of $C_k(X_p(\omega))$ and $e_b(X_p(\omega))$ denotes the edge length of the buckets in the neighborhood. However, the edge length of a hypercube is the d-th root of its volume, hence

$$e_k(X_p(\omega)) = \sqrt[d]{V_k(X_p(\omega))}, \qquad e_b(X_p(\omega)) = \sqrt[d]{V_b(X_b(\omega))}.$$

Assume that buckets around $B_k(X_p(\omega))$ are smaller than $B_k(X_p(\omega))$ and since they are close, $\bar{p}(X_p(\omega)) \approx \bar{p}(X_b(\omega))$. Then, by (2.5) we have

$$e_k(X_p(\omega)) \approx \sqrt[d]{\frac{k \cdot G(d)}{(n+1)\bar{p}(X_p(\omega))}}, \qquad e_b(X_p(\omega)) = \sqrt[d]{\frac{b \cdot G(d)}{(n+1)\bar{p}(X_b(\omega))}}.$$

Finally,

$$\bar{l} \leq \bar{L} = \left(\left\lfloor \frac{e_k(X_p(\omega))}{e_b(X_p(\omega))} \right\rfloor + 1 \right)^d \approx \left(\left\lfloor \frac{\sqrt[d]{\frac{k \cdot G(d)}{(n+1)\bar{p}(X_p(\omega))}}}{\sqrt[d]{\frac{b \cdot G(d)}{(n+1) \cdot \bar{p}(X_b(\omega))}}} \right\rfloor + 1 \right)^d$$
$$= \left(\left\lfloor \sqrt[d]{\frac{k}{b}} \right\rfloor + 1 \right)^d \leq \left(\left(\frac{k}{b} \right)^{\frac{1}{d}} + 1 \right)^d$$

is an upper bound for the average number of buckets overlapping $B_k(X_p(\omega))$. Here, we use that $B_k(X_p(\omega))$ is small and thus we have $X_p(\omega) \approx X_b(\omega)$. Note that the inequality holds as $\sqrt[d]{k/b} \geq 0$. The number of records in each bucket is b, so an upper bound on the number of records examined is

$$\bar{R} \leq b \cdot \bar{L} \leq b \cdot \left(\left(\frac{k}{b} \right)^{1/d} + 1 \right)^d = \left(k^{1/d} + b^{1/d} \right)^d .$$

This expression is minimized when choosing $b = 1$ (which explains the corresponding choice in Section 2.1), yielding

$$\bar{R} \leq \left(k^{1/d} + 1 \right)^d , \tag{2.6}$$

which is independent of the number of records n and the density $\delta(x)$.

The constancy of the number of records examined as the number of records increases implies that the time required for a nearest neighbor search is equal to finding the record in the balanced binary tree, the k-d tree, which takes $\mathcal{O}(\log(n))$ on average. Thus, the following theorem is proven:

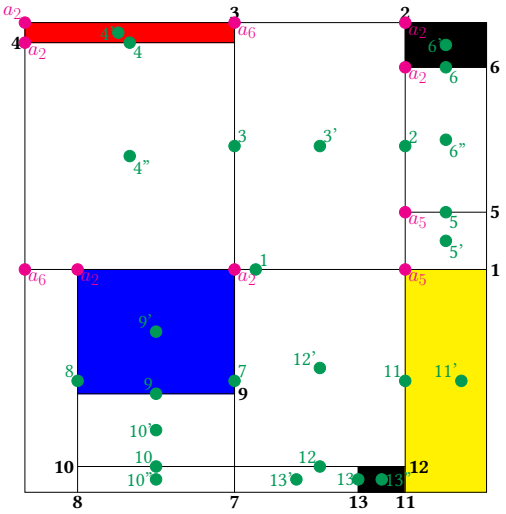
Theorem 2 (Runtime of nearest neighbor queries in a k-d tree, [FBF77])**.** *The expected query time for a neighborhood query in a k-d tree is* $\mathcal{O}(\log(n))$.

2.3 Conclusion and Addendum: k-d Trees in Arts

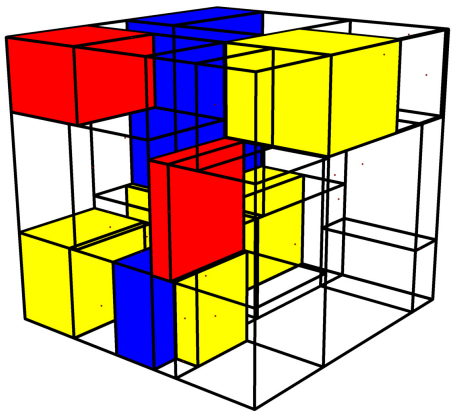
We have presented the data structure of k-d trees and have given the classical proof of Friedman, Bentley, and Finkel in a modernized form. The proof shows the practical relevance of k-d trees for the field of geometry processing, as all neighborhood concepts presented in Section 1 can be computed efficiently utilizing the k-d tree data structure.

Additionally, k-d trees have surprising applications aside from geometry processing. We have recently shown that they can be used to automatically generate art pieces that bear a striking resemblance to those of the dutch artist Piet Mondrian. Furthermore, as k-d trees are not limited to two-dimensional data, they provide a mean to create three-dimensional "Mondrian-like" structures. See Figure 2.3a for a painting of Piet Mondrian, Figure 2.3b for a reproduction of this paining by a two-dimensional k-d tree, and Figure 2.3c for a three-dimensional "Mondrian-like" structure. Further details on this work, as well as other generalization of artworks from two to three dimensions can be found in the works published at the "Bridges" conference, see the list of publications prior to the thesis, page 5.

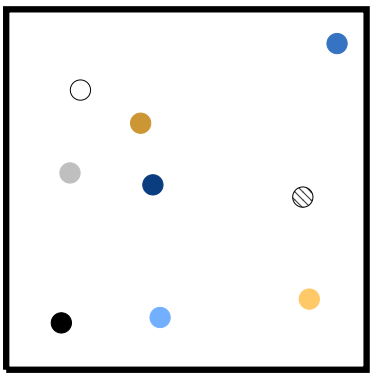
(a) *Piet Mondrian, "Tableau I", 1921, oil on canvas, Collection Gemeentemuseum Den Haag.*



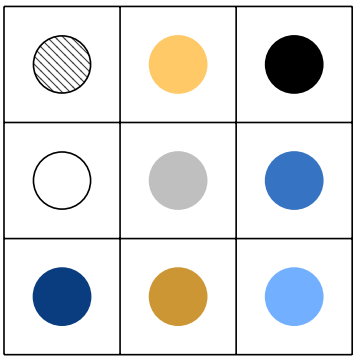
(b) *Reproduction of "Tableau I" utilizing a k-d tree.*



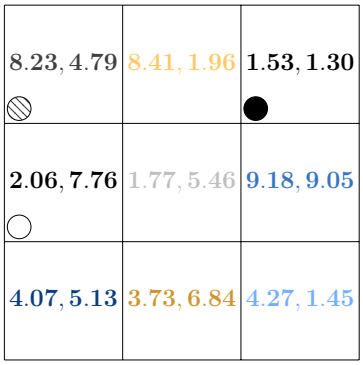
(c) *Visualization of a three-dimensional k-d tree on 43 points, colored as a "Mondrian-like" structure.*



(a) *Point set with nine points in* $\mathbb{R}^2$.



(b) *The points are initially placed randomly.*



(c) *The point coordinates are considered.*

Figure 3.1: *First part of the neighborhood grid pipeline.*

3 The Neighborhood Grid

In the previous section, we have discussed the data structure of k-d trees to compute results to neighborhood queries. As proven, it works well in practice on a single-core system, but does not benefit from parallelization. With the wide availability of graphic cards and computing clusters, this application becomes more and more important. Thus, in this section, we turn to a different data structure: the neighborhood grid. In contrast to k-d trees, the answers to neighborhood queries computed by the neighborhood grid are not always exact, but can be easily parallelized and can thus be computed faster.

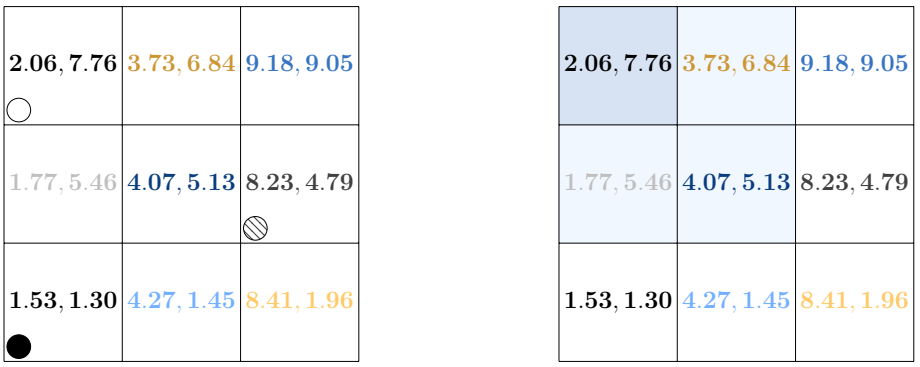
We collect new results on the neighborhood grid data structure. The data structure has been introduced by Joselli et al., see [Jos+09; Jos+15]. Malheiros and Walter [MW15] investigated several iterative building strategies for the data structure. Despite the evidences of practical relevance, as demonstrated in the publications cited above, Joselli et al. did not investigate the asymptotic building times of the grid. Malheiros and Walter gave a proof for time-optimality of a building algorithm, which, however, does contain a flaw. Therefore, the main contributions of this chapter are:

- Proof of asymptotic time-optimality of a presented building algorithm (Theorem 5).
- Comparison of the single-core building algorithm with the parallel building algorithm of Malheiros and Walter [MW15] (Section 3.5.2).
- Combinatorial results on the number of possible sorted placements (Theorem 4).
- A complete list of unique sorted placements for $n \in \{1, 2, 3\}$ (Section 3.3.3).
- A proof of non-existence of unique sorted placements for $n \geq 4$ (Section 3.3.3).
- Results on the neighborhood quality obtained from the neighborhood grid (Section 3.6).

Finally, we present a conjecture (Conjecture 1) on a part of the following open question:

- For a given $n \in \mathbb{N}$, $n \geq 4$, what is a point set with the least or largest number of stable states?

The results of this section are available on ArXiv and have been presented at the EuroCG18 conference, see list of publications prior to the thesis, page 5. Additional to the content in the publications, this section contains elaborate combinatorial results, illustrations of the unique stable states, and experiments on the neighborhood quality.



(a) *Coordinates are sorted to grow in columns and rows.*

(b) *Determining the neighbors of the* ***upper left point*** *by looking at its* ***neighbors*** *in the grid.*

Figure 3.2: *Second part of the neighborhood grid pipeline.*

3.1 Introduction of the Data Structure and a Polynomial Time-Optimal Building Algorithm

3.1.1 A Short Informal Introduction to the Neighborhood Grid

In order to give a short introduction to the data structure, consider the example in Figure 3.1. It shows how points from a point set (Figure 3.1a) are placed in a grid (Figure 3.1b). The order in which the points are given is random, thus their initial placement in the grid is also. After the placement, only the coordinates of the points are considered in the grid (Figure 3.1c).

The grid as obtained in Figure 3.1c will now be sorted. Each row should grow in the first coordinates from left to right, each column should grow in the second coordinates from bottom to top. A corresponding sorted grid is given in Figure 3.2a. Note how it—in this example—recovers the combinatorial neighborhood relation from the points.

In order to use the neighborhood grid to determine a neighborhood estimate for a given point, find that point in the sorted grid. Then, consider a small neighborhood around that point, e.g. the one-ring around it. The size of this neighborhood should not depend on the number of inserted points such that the time of this lookup only depends on the dimension and is asymptotically constant for some fixed dimension d. From that neighborhood, find the closest point to the considered point and output it as estimated nearest neighbor, see Figure 3.2b.

3.1.2 Definition of the Data Structure

Given a set of points $P = \{p_1, \dots, p_N \mid p_i \in \mathbb{R}^d\}$. In the following we will assume that $N = n^2$ for some $n \in \mathbb{N}$ and $d = 2$. Therefore each point is given by $p_i = (p_i^1, p_i^2) \in \mathbb{R}^2$, where p_i^1 will be referred to as x- and p_i^2 as y-value of the points p_i. Furthermore, we assume that $p_i \neq p_j$ for all $i \neq j$. Consider Section 3.5.1 for the general case without these restrictions.

$(55, 42)$	$(60, 82)$	
	$(26, 61)$	$(13, 69)$
$(95, 13)$	$(95, 10)$	

$(06, 69)$	$(26, 61)$	$(86, 89)$
$(02, 55)$	$(80, 34)$	$(86, 41)$
$(05, 19)$	$(47, 11)$	$(95, 13)$

Figure 3.3: *On the left three cases where two marked in blue satisfy the first condition of Definition 3, while the yellow case violates it. On the right a* 3×3 *matrix* $M_\pi(P)$ *in stable state.*

Definition 2. *Given a set of points* P *as specified above, a* placement $\pi : [n^2] \to [n] \times [n], i \mapsto (k, \ell)$ *is a bijective map such that the matrix* $M_\pi(P)$ *is given as*

$$M_\pi(P) = \begin{array}{|c|c|c|} \hline (p^1_{\pi^{-1}(n,1)}, p^2_{\pi^{-1}(n,1)}) & \cdots & (p^1_{\pi^{-1}(n,n)}, p^2_{\pi^{-1}(n,n)}) \\ \hline \vdots & \ddots & \vdots \\ \hline (p^1_{\pi^{-1}(1,1)}, p^2_{\pi^{-1}(1,1)}) & \cdots & (p^1_{\pi^{-1}(1,n)}, p^2_{\pi^{-1}(1,n)}) \\ \hline \end{array}. \tag{3.1}$$

Ultimately, we want to order the points in the matrix such that the following state is reached.

Definition 3. *The matrix* $M_\pi(P)$ *as given in (3.1) is said to be in a* stable state, *respectively the placement* π *of Definition 2 is* stable, *if and only if the following two conditions are satisfied for any* $i, j \in [n], i \neq j$.

1. *For all* $k \in [n]$ *it is:*
$$i < j \Rightarrow p^1_{\pi^{-1}(k,i)} < p^1_{\pi^{-1}(k,j)} \vee \left(p^1_{\pi^{-1}(k,i)} = p^1_{\pi^{-1}(k,j)} \wedge p^2_{\pi^{-1}(k,i)} < p^2_{\pi^{-1}(k,j)}\right).$$
2. *For all* $\ell \in [n]$ *it is:*
$$i < j \Rightarrow p^2_{\pi^{-1}(i,\ell)} < p^2_{\pi^{-1}(j,\ell)} \vee \left(p^2_{\pi^{-1}(i,\ell)} = p^2_{\pi^{-1}(j,\ell)} \wedge p^1_{\pi^{-1}(i,\ell)} < p^1_{\pi^{-1}(j,\ell)}\right).$$

Note that these conditions are well-defined as we assumed $p_i \neq p_j$. *See Section 3.5.1 for the general case.*

In other words, a matrix $M_\pi(P)$ is in a stable state, if the points in each row of M are ordered lexicographically according to the first and then the second coordinate. Also, all columns of M have to be ordered lexicographically according to the second and then the first coordinate. An illustration of Definition 3 is given in Figure 3.3. We call a stable state *unique*, if there exists no other stable state for the same point set P.

3.1.3 Polynomial-Time Building Algorithm

Now the following question arises naturally: for any set of points P as specified above, can we find a stable placement? In other words, given n^2 points, can these be written into an $n \times n$ matrix s.t. it is in a stable state as defined in Definition 3. Indeed, Malheiros and Walter [MW15] gave an algorithm to create a stable state from any given point set P. We state it here as a theorem.

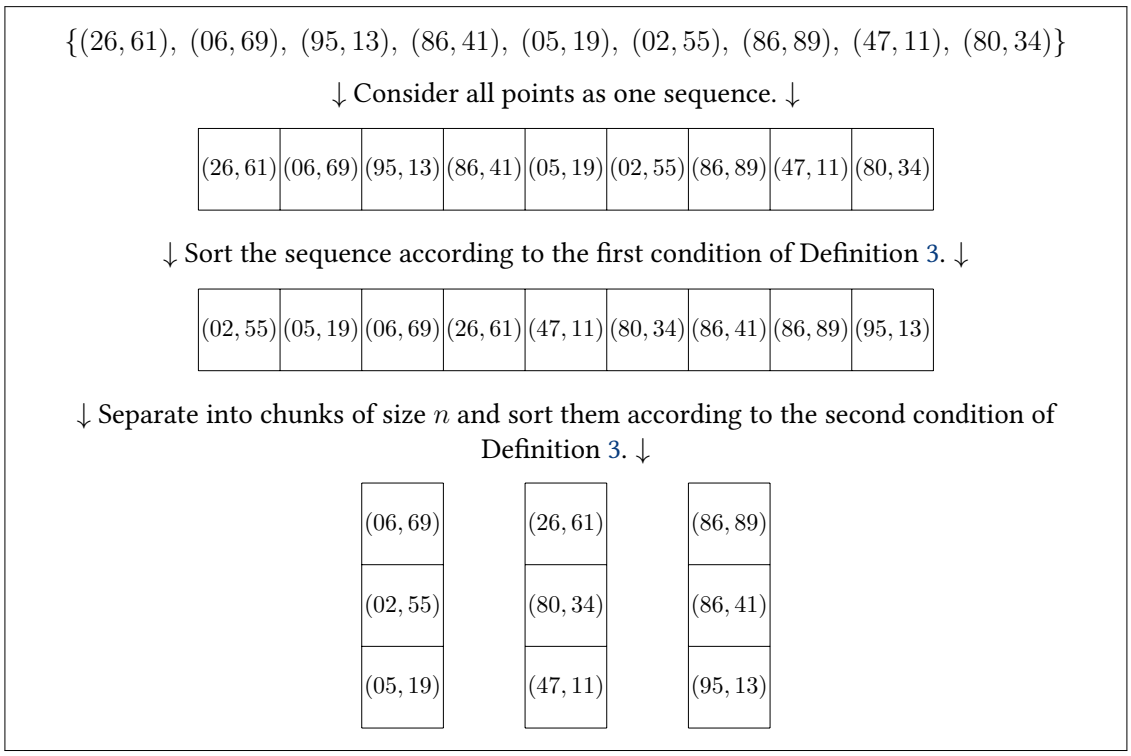


Figure 3.4: *An illustration of the algorithm outlined in Theorem 3. The last step gives the rows of the final matrix which is then in a stable state.*

Theorem 3 (Construction of a stable sate, [MW15])**.** *For every set of points* $P = \{p_1, \dots, p_N \mid p_i \in \mathbb{R}^2\}$ *there is a stable placement* π*. Such a stable placement can be found in at most* $\mathcal{O}(N \log(N))$ *when considering a comparison-based setting*[9]*.*

Proof. Consider the points $p_1, \dots, p_N$ as a sequence. Sort this sequence according to the first condition given in Definition 3. Obtain a sequence

$$(q_1^1, q_1^2), (q_2^1, q_2^2), \dots, (q_N^1, q_N^2),$$

where for $i, j \in [N]$, $i < j$ we have $q_i^1 < q_j^1$ or $(q_i^1 = q_j^1 \wedge q_i^2 < q_j^2)$. Now split this sequence into n blocks of size n as follows:

$$\underbrace{(q_1^1, q_1^2), \dots, (q_n^1, q_n^2)}_{=:Q_1}, \underbrace{(q_{n+1}^1, q_{n+1}^2), \dots, (q_{2n}^1, q_{2n}^2)}_{=:Q_2},$$
$$\dots \underbrace{(q_{n^2-n+1}^1, q_{n^2-n+1}^2), \dots, (q_N^1, q_N^2)}_{=:Q_n}.$$

Now consider each sequence Q_i and sort it according to the second condition given in Definition 3. Obtain a sequence

$$R_k := (r_1^1, r_1^2), (r_2^1, r_2^2), \dots, (r_n^1, r_n^2),\ k \in [n],$$

[9] Throughout the remainder of this section, we will assume a comparison-based model. That is, when sorting keys, we assume that information on a pair of keys is given solely by a comparison running in constant time. This implies that we neglect effects of the bit length of the real numbers considered. In more general settings, sorting of n integers can be achieved in $\mathcal{O}(n \log(\log(n)))$, see [And+98].

where for $i, j \in [n]$, $i < j$ we have $r_i^2 < r_j^2$ or $(r_i^2 = r_j^2 \wedge r_i^1 < r_j^1)$. That is, the points in the sequence R_k are sorted according to the second condition of Definition 3. Furthermore, for $i, j \in [n]$, $i < j$, any point from R_i satisfies the first condition of Definition 3 when compared to any point from R_j, since the R_k are derived from the Q_k. Therefore, placing the sequence R_k into the kth column of the matrix M results in a stable state.

Concerning the runtime, in the first step, N points were sorted, which is well known to take $\mathcal{O}(N \log(N))$. In the second step, n sets of n points each were sorted, which takes

$$n \cdot \mathcal{O}(n \log(n)) = \mathcal{O}(n^2 \log(\sqrt{N})) = \mathcal{O}(N \log(N))$$

Hence, the stable state was computed in $\mathcal{O}(N \log(N))$. □

An illustration of the procedure presented by this theorem is given in Figure 3.4.

Theorem 3 imposes an upper bound on the runtime of any time-optimal comparison-based algorithm that creates a stable state of a matrix $M_\pi(P)$. The next question is then: What is a lower bound? At this point, Malheiros and Walter [MW15] state:

> *Note that the problem of sorting n unrelated lists of n real values has $\mathcal{O}(n^2 \log n^2)$ as its established lower bound. We can build an associated spatial sorting problem by copying each unsorted list to a matrix row, setting its x coordinates. We can also define that for the i-th row, all its y coordinates will be set to i. If the matrix is then spatially sorted using the algorithm described earlier, the lists will be ordered. Therefore, $\mathcal{O}(N \log N)$ is also a lower bound for spatial sorting.*

The flaw in this argument is that the grid will not necessarily come to a stable state in which the described n lists are placed in the respective rows. That is due to the fact that a given point set can have multiple stable states which we will show in the following.

3.2 Combinatorial Results on Stable States of the Neighborhood Grid

3.2.1 Counting Stable States

Before we proceed to deduce a lower bound on the building time of a neighborhood grid, we first give some combinatorial results on the number of stable states. Concerning these, it is combinatorially not important what actual x- or y-values the points have, but only how these values compare to each other. Therefore, the setup can be simplified by fixing the x and y values each on the set $[N]$. All combinations are given by the following setup: w.l.o.g. let $p_i^1 = i$ for $i \in [N]$. Then, P is fully determined by a permutation $\sigma \in \mathbb{S}_N$, where $P = \{(i, \sigma(i)) \mid i = 1, \ldots, N\}$. On these restricted point sets, we can make the following statements.

Theorem 4 (Counting stable states, U. Reitebuch and M. S.)**.** *Given some $N, n \in \mathbb{N}$ with $N = n^2$, there are:*

1. $(n^2)!$ *restricted point sets P and thereby $((n^2)!)^2$ different ways to fill the matrix,*
2. $((n^2)!/(n!)^n)^2$ *ways to fill the matrix with a stable state,*
3. *each placement π is stable for exactly $(n^2)!/(n!)^n$ restricted point sets P,*
4. $1/(n!)^{2n}$ *of all fillings of the matrix are stable.*

Proof. Concerning the first statement, when building the points p_i, iterate through the x-values. For each x-value pick one of the y-values from $[N]$ not picked yet. Then for $x = 1$, there are N choices, for $x = 2$, there are $N - 1$ choices, etc. Hence there are $N!$ possible point sets. Each p_i of the point set P occupies one of the N points in the matrix, therefore, there are $N!$ ways to write each point set into the matrix. Therefore, there are $(N!)^2$ possible ways to fill the matrix.

Concerning the second statement, a matrix is in a stable state if both conditions of Definition 3 are met. Because of the restriction on the point sets as declared above, the equality case is never met in either condition of Definition 3. Therefore, for each condition, it suffices to check the x- or y-values respectively, neglecting the other. When setting up the x-values for the first row, one can pick n of the possible N values, which then admit to a unique order. Therefore, for the x-values in the first row, there are $\binom{n^2}{n}$ possibilities. For the second row, there are $\binom{n^2-n}{n}$ possibilities, until there is $\binom{n^2-(n-1)n}{n} = 1$ possibility for the last row. Overall, there are

$$\prod_{k=0}^{(n-1)} \binom{n^2 - kn}{n} = \frac{n^2!}{(n^2-n)!n!} \cdot \frac{(n^2-n)!}{(n^2-2n)!n!} \cdot \ldots \cdot \frac{(2n)!}{n!n!} \cdot \frac{n!}{n!} = \frac{(n^2)!}{(n!)^n}$$

possibilities to put x-values into the matrix and obtain a stable state from them. Accordingly, there are $\frac{(n^2)!}{(n!)^n}$ ways to write y-values into the matrix and obtain a stable state. Hence, overall, there are

$$\left(\frac{(n^2)!}{(n!)^n}\right)^2$$

stable states.

By a similar argument, we can compute the number of point sets P for which a given placement π is stable. Consider a placement π that fixes the x-values in $M_\pi(P)$ such that they are stable. When counting the number of point sets for which π is stable, we can now pair the already placed x-values with y-values as follows: When setting up the y-values for the first column, one can pick n of the possible $N = n^2$ values, which then admit to a unique order in the column. Therefore, for the y-values in the first column, there are $\binom{n^2}{n}$ possibilities, for the second column, there are $\binom{n^2-n}{n}$ possibilities, etc. until overall, there are $\frac{(n^2)!}{(n!)^n}$ possibilities to put y-values into the matrix and obtain a stable state from them utilizing the fixed π. That is, a placement π is always stable for exactly $\frac{(n^2)!}{(n!)^n}$ point sets.

Finally, because of the first two results, the fraction of stable fillings amongst all fillings of the matrix are

$$\frac{\left(\frac{(n^2)!}{(n!)^n}\right)^2}{((n^2)!)^2} = \frac{1}{(n!)^{2n}}.$$

□

3.2.2 Lower Bound

Having the necessary results at hand, we can now prove a lower bound on the building time of the neighborhood grid. Consider any comparison-based algorithm $\mathcal{A}$ that creates a stable placement for a given point set. Each query of $\mathcal{A}$ can be considered as a node of a decision-tree where the leaves correspond to placements of which some are stable for the given set. For an optimal algorithm, this tree is balanced and has depth $\log((n^2)!)$. Recall the result of Theorem 4 that any

placement π is stable for $\frac{(n^2)!}{(n!)^n}$ restricted point sets. Thus, when building the tree, the algorithm $\mathcal{A}$ cannot stop at a subtree with more than $\frac{(n^2)!}{(n!)^n}$ leaves, as one of them will surely not be stable under the currently considered placement. That is, the tree has to be traversed to depth at least

$$\log((n^2)!) - \log\left(\frac{(n^2)!}{(n!)^n}\right) = \log\left(\frac{(n^2)! \cdot (n!)^n}{(n^2)!}\right) = \log((n!)^n) = n \cdot \log(n!)$$
$$\in \Omega(n^2 \cdot \log(n)).$$

Therefore, each comparison-based algorithm building a stable state needs to perform at least $\Omega(n^2 \cdot \log(n))$ operations. This proves the following theorem:

Theorem 5 (Time-optimality of the building algorithm, S. Das, U. Reitebuch, and M. S.)**.** *The algorithm presented in the proof of Theorem 3 is a time-optimal building algorithm for the neighborhood grid amongst all comparison-based algorithms and takes $\Theta(N \log(N))$ time to build a grid for a point set P of N points.*

3.3 Uniqueness of Stable States

In the previous section it was shown that the algorithm outlined in Theorem 3 is a time-optimal building algorithm for the neighborhood grid. In their article [MW15], Malheiros and Walter give a different proof for the same fact. We will present the proof here and show in the upcoming sections why it is flawed for larger n. First, we make the following assumption.

Assumption 1. *Given some $n \in \mathbb{N}$, assume that there is a set P of points $p_1, \ldots, p_N$ such that there exists a unique stable state for P.*

Given that assumption, optimality of the Algorithm of Theorem 3 follows by the following argument as given by Malheiros and Walter [MW15]:

Proof. Given an $n \in \mathbb{N}$ and the set of points P assumed to exist by Assumption 1. Then, the unique stable state in particular satisfies the first condition of Definition 3 for every row. Therefore, the algorithm needs to sort every row, which takes at least $\Omega(n \log(n))$ time for each row, that is $n \cdot \Omega(n \log(n)) = \Omega(N \log(N))$ time at least in total. Hence, given Assumption 1, $\Omega(N \log(N))$ is a lower bound on the computation of a stable state. □

Note that the assumption of the existence of a unique stable state is crucial in this argument. If the assumption was proven to be wrong, the algorithm could pick any stable state to create a sorting of the rows there, which could lead to a faster than $\Theta(N \log(N))$ algorithm because of more searched-for elements in an equally large search space.

Concerning Assumption 1, it is not at all obvious that for a given point set, there has to be a unique stable state. Three point sets are given in Figure 3.5, where two have a unique stable state, while the other one has two stable states. The uniqueness of the stable state of the 2×2 matrix shown follows by this reasoning: The element $(1, 1)$ has to be placed in the lower left corner, since it has lowest x-, as well as lowest y-value compared to all other elements. The element $(2, 4)$ has largest y-value, therefore it needs to be placed in the upper row. But it also has second lowest x-value, therefore it has to be placed in the upper left or lower right corner. Taken these two arguments together, the element has to be placed in the upper left corner. Finally, the shown set is the only way to complete the matrix to a stable state.

The examples from Figure 3.5 pose two further questions:

$(2,4)$	$(3,3)$
$(1,1)$	$(4,2)$

$(2,2)$	$(4,4)$
$(1,1)$	$(3,3)$

$(3,3)$	$(4,4)$
$(1,1)$	$(2,2)$

$(3,7)$	$(4,8)$	$(9,9)$
$(2,6)$	$(5,5)$	$(8,4)$
$(1,1)$	$(6,2)$	$(7,3)$

Figure 3.5: *On the left a point set with $n = 2$ which has a unique stable state. In the middle a point set with $n = 2$ with its two possible stable states. On the right a point set with $n = 3$ which has a unique stable state.*

$(2,3)$	$(4,4)$
$(1,1)$	$(3,2)$

$(3,2)$	$(4,4)$
$(1,1)$	$(2,3)$

Figure 3.6: *On the left hand side a set in stable state, satisfying both x- and y-bin condition. On the right a stable state of the same set P, showing that x- and y-bin condition do not imply uniqueness.*

1. Is there a restricted point set P for every $n \in \mathbb{N}$ that has a unique stable state? That is, does Assumption 1 hold?
2. Can we classify all restricted point sets with unique stable states?

3.3.1 First Necessary Condition on Stable States

For a necessary condition on the uniqueness of a stable state, reconsider the algorithm outlined in Theorem 3. Note that the n smallest x-values are placed in the first column, the n second-smallest x-values are placed in the second column, etc. Finally, a stable state is obtained from this procedure. Furthermore, the algorithm can be run the other way around, placing the n smallest y-values in the first row, the n second-smallest y values in the second row, and so on to reach a stable state. In the following, we will call these two conditions *x-bin and y-bin condition.*

Assume now that a stable state is given, which violates the x-, the y-, or both bin conditions. Then, applying the algorithm from Theorem 3 with sorting according to the violated bin condition gives another stable state for the same points. This observation establishes the following necessary condition for the uniqueness of a stable state.

Theorem 6 (First necessary condition on stable states, U. Reitebuch and M. S.)**.** *Given a matrix M in stable state. If the stable state is unique, then both the x- and the y-bin condition hold.*

Note that Theorem 6 does not give an equivalence. Consider the point set

$$P = \{(1,1), (2,3), (3,2), (4,4)\}.$$

A stable state of this set satisfying both x- and y-bin condition is shown on the left side of Figure 3.6. However, the point set admits to a second stable state, shown on the right side of Figure 3.6. This showcases that x- and y-bin condition are only necessary, but not sufficient conditions.

$$
\begin{array}{c}
 \\
(n!)^n \\
((n^2)!/(n!)^n) - (n!)^n \\
(n^2)! - ((n^2)!/(n!)^n)
\end{array}
\begin{array}{c}
\begin{array}{ccc}
(n!)^n & ((n^2)!/(n!)^n) - (n!)^n & (n^2)! - ((n^2)!/(n!)^n)
\end{array} \\
\left(\begin{array}{ccc}
\text{stable } x-y-\text{bin} & \text{stable } x-\text{bin} & \text{not stable} \\
\text{stable } y-\text{bin} & \text{stable} & \text{not stable} \\
\text{not stable} & \text{not stable} & \text{not stable}
\end{array}\right)
\end{array}
$$

Figure 3.7: *Distribution of possibly unique stable states amongst all stable states and amongst all states. The outer left column show the number of states satisfying the indicated conditions on the x-values while the top row shows the number of states satisfying the indicated condition on the y-values.*

n	stable states $(\frac{(n^2)!}{(n!)^n})^2$	x-y-bin stable $(n!)^{2n}$
1	1	1
2	36	16
3	2,822,400	46,656
4	3,976,941,969,000,000	110,075,314,176

Table 3.1: *A numerical comparison of the number of stable states $(\frac{(n^2)!}{(n!)^n})^2$ and the number of x-y-bin stable states $(n!)^{2n}$.*

Theorem 4 already tells us that there are $(n^2)!/(n!)^n$ ways to put x-values into the matrix such that they form a stable state. Furthermore, there are $(n^2)!$ possible ways to fill the matrix with x-values. How many of these satisfy the condition of Theorem 6?

Theorem 7 (Upper bound on number of unique stable states, U. Reitebuch and M. S.)**.** *There are $(n!)^{2n}$ stable states that satisfy both the x- and the y-bin condition.*

Proof. Assume the x- and y-bin condition holds, then the first row contains the n smallest y-values, while the first column contains the n smallest x-values. Either values can be permuted arbitrarily without destroying the stable state. This holds for every row and every column. Therefore, there are $(n!)^n$ ways to reorganize the rows and similarly $(n!)^n$ ways to reorganize the columns and still obtain a stable state. Hence, $(n!)^{2n}$ stable states satisfy both bin conditions. □

The situation established in Theorems 4 and 7 can be visualized as in Figure 3.7. A numerical comparison of the number of stable states $(\frac{(n^2)!}{(n!)^n})^2$ and the number of x-y-bin stable states $(n!)^{2n}$ is given in Table 3.1.

However, we can derive even more from the algorithm of Theorem 3.

3.3.2 Second Necessary Condition on Stable States

Given the terminology and results from Section 3.3.1, we can derive an even stronger necessary condition for unique stable states. But before stating and proving it, we state the following corollary which can be directly derived from Definition 3.

Corollary 1 (Submatrices of stable states are stable, U. Reitebuch and M. S.)**.** *Given an $n \times n$ matrix $M_\pi(P)$ in a stable state, then all $\ell \times \ell$ submatrices of $M_\pi(P)$ with $\ell \in [n]$ are also in a stable state.*

Utilizing this corollary, we can now make the following stronger statement on submatrices of matrices in a unique stable state.

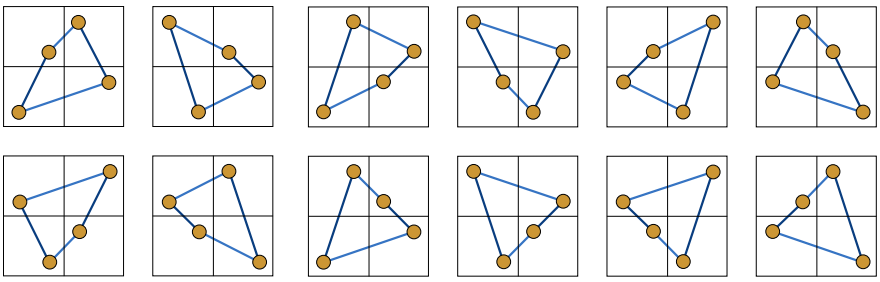


Figure 3.8: *All* 12 *unique* 2×2 *stable states. Light blue edges correspond to horizontal neighborhood relation in the grid, while dark blue edges indicate a vertical neighborhood relation in the grid.*

Theorem 8 (Second necessary condition on stable states, U. Reitebuch and M. S.)**.** *Given an* $n \times n$ *matrix* $M_\pi(P)$ *in stable state. If the stable state is unique, then any* $\ell \times \ell$ *connected submatrix of* $M_\pi(P)$ *with* $\ell \in [n]$ *is in a unique stable state.*

Proof. Given some $n \times n$ matrix $M_\pi(P)$ in a unique stable state. Assume there exists some $\ell \times \ell$ connected submatrix $\widetilde{M}$ of M, $\ell \in [n]$, such that $\widetilde{M}$ does not have one unique stable state, but has a different stable state $\overline{M}$. Assume that $\widetilde{M}$ occupies rows $r, \ldots, r + (\ell - 1)$ and columns $c, \ldots, c + (\ell - 1)$ in M, with $r, c \in [n - \ell]$.

Because of the first necessary condition on stable states (Theorem 6), we know that the x-values of any elements in columns $1, \ldots, c-1$ are smaller and the x-values of any elements in the columns $c+\ell, \ldots, n$ are larger than all x-values in $\widetilde{M}$ respectively. This remains true independent of any stable reordering within the submatrix, in particular for the reordering induced by $\overline{M}$. The same argument holds for the y-values in rows $1, \ldots, r-1$ and the y-values in rows $r+\ell, \ldots, n$ when compared with the y-values of $\widetilde{M}$ and $\overline{M}$ respectively. Therefore, replacing $\widetilde{M}$ by $\overline{M}$ in M gives another stable state, which violates the uniqueness of M. □

3.3.3 Enumeration of all unique Stable States

The number of unique stable states for any $n \in \mathbb{N}$ is given in Table 3.2. All unique stable states for $n = 2$ and $n = 3$ are shown in Figures 3.8 and 3.9 respectively. Brute-Force computation shows that none of the $37,536$ possible 4×4 point sets satisfying the second necessary condition established in Theorem 8 is uniquely stable. That is, Assumption 1 does not hold. Furthermore, from $n = 4$ upward, there can never again be any uniquely stable point set, as if there was one, it would have to include a unique stable 4×4 state by Corollary 1, which does not exist. This proves the argument of Malheiros and Walter [MW15] to be incorrect.

The currently lowest number of states for $N = 16$ is 7 and was found by Dr. Adrian Neumann. The fact that for $n \geq 4$ there is no point set with a unique stable state raises the following question:

Open Question 1. *Given* $n \in \mathbb{N}$, $n \geq 4$, *what is a point set* P *with the minimum number of stable states among any point set with* n^2 *points?*

n	1	2	3	≥ 4
# stable states	1	12	966	0

Table 3.2: *Number of unique stable states for* $n \times n$ *matrices.*

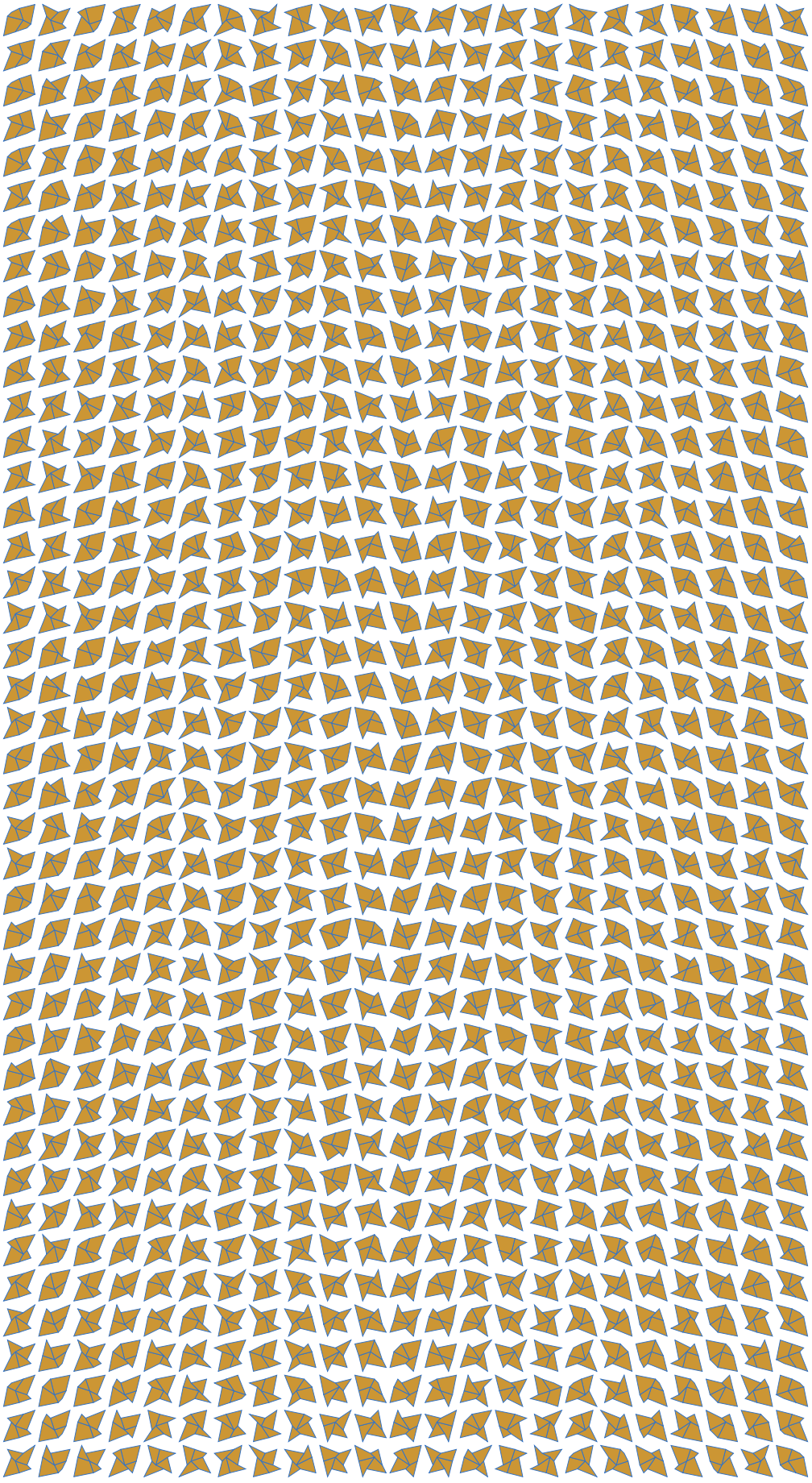


Figure 3.9: *All* 966 *unique* 3×3 *stable states.*

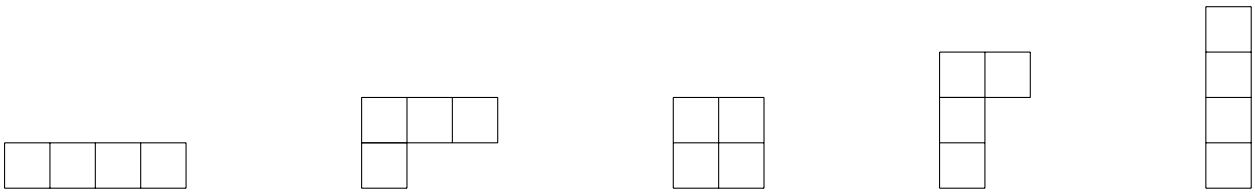


Figure 3.10: *The five possible Ferrers diagrams for the partitions of* $N = 4$ *given by* $\lambda = (4)$, $\lambda = (3,1)$, $\lambda = (2,2)$, $\lambda = (2,1,1)$ *and* $\lambda = (1,1,1,1)$.

1	2	3
4		

4	1	3
2		

Figure 3.11: *Two Young tableaux corresponding to the partition* $\lambda = (3,1)$ *of* $N = 4$. *The left tableau is standard, while the right one is not.*

3.4 The worst Stable State

We proceed by turning the question from the last paragraph around. What is the maximal number of stable states a point set can obtain for some given $n \in \mathbb{N}$? In order to investigate this question, we first turn to a specific point set, for which we can count the number of stable states. Consider the "diagonal[10]": $\{(1,1),(2,2),\ldots,(n^2,n^2)\}$. Counting the number of stable states for the diagonal is equivalent to placing only one number in each field of the $n \times n$ matrix, which then has to satisfy both conditions of Definition 3.

In order to count the number of stable states for the diagonal, we introduce the concept of Ferrers diagrams and Young tableaux reproduced from [Sag01].

Definition 4. *Let* $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_\ell)$, $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_\ell$ *be a partition of* N. *The* Ferrers diagram *of* λ *is an array of* N *cells having* ℓ *left-justified rows with row* i *containing* λ_i *cells for* $1 \leq i \leq \ell$.

Consider for example $N = 4$. Then, N can be partitioned in five different ways:

$$4 \qquad = 3+1 \qquad = 2+2 \qquad = 2+1+1 \qquad = 1+1+1+1,$$

as illustrated in Figure 3.10.

Definition 5. *Let* $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_\ell)$, $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_\ell$ *be a partition of* N. *A* Young tableau *of shape* λ *is an array* t *obtained by filling the cells of the Ferrers diagram of* λ *with the numbers* $1, 2, \ldots, N$ *bijectively. A tableau* t *is* standard *if the rows and columns of* t *are increasing sequences.*

See Figure 3.11 for an illustration of this definition.

Clearly, the number of standard Young tableaux of shape $\lambda = (n, \ldots, n)$ for $N = n^2$ is equal to the number of stable states for the diagonal. Denote by f^λ the number of standard λ-tableaux. In order to compute it, we introduce the concept of hooks.

Definition 6. *If* $v = (i,j)$ *is a node in the Ferrers diagram of* λ, *then it has* hook

$$H_v = H_{i,j} = \{(i,j') \mid j' > j\} \cup \{(i',j) \mid i' \geq i\}$$

with corresponding hook-length

$$h_v = h_{i,j} = |H_{i,j}|.$$

[10] We favor this name alluding to the diagonal morphism in category theory. However, it is the identity permutation and is therefore referred to as "identity" in the corresponding publications listed on page 5.

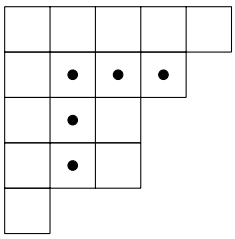


Figure 3.12: *Given the partition $\lambda = (5, 4, 3, 3, 1)$ of $N = 16$, the dots show the hook $H_{2,2}$ with a hook length of $h_{2,2} = 5$.*

See Figure 3.12 for an illustration of this definition. It is now easy to state the hook formula of Frame, Robinson, and Thrall, see [Sag01, p. 124].

Theorem 9 (Hook formula, [Sag01], p. 124)**.** *Let $\lambda = (\lambda_1, \ldots, \lambda_\ell)$, $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_\ell$ be a partition of N. Then*

$$f^\lambda = \frac{N!}{\prod_{(i,j)\in\lambda} h_{i,j}}.$$

In the concrete case of the diagonal, where $N = n^2$, $\lambda = (n, \ldots, n)$, we have

$$h_{i,j} = n - i + n - j + 1 = 2n - i - j + 1.$$

Therefore, the number of stable states of the diagonal is given by

$$f^{(n,\ldots,n)} = \frac{N!}{\prod_{i=1}^{n}\prod_{j=1}^{n}(2n - i - j + 1)}. \tag{3.2}$$

The results for $n \in \{1, 2, 3\}$ and computational experiments lead us to state the following conjecture.

Conjecture 1 (Upper bound stable state, U. Reitebuch and M. Skrodzki)**.** *Given $n \in \mathbb{N}$, the number of stable states of any point set P on n^2 points is less or equal to $f^{(n,\ldots,n)}$.*

The number $f^{(n,\ldots,n)}$ also counts the number of linear extensions of the $n \times n$ lattice and is thereby connected to posets (partially ordered sets). This gives another angle at Conjecture 1 as we will elaborate in the following.

Definition 7. *Given a set A and a symmetric, anti-symmetric, and transitive relation $\succeq$ on the elements of A. Then $(A, \succeq)$ is called a* partially ordered set*, short:* poset*.*

The specific entities that we want to relate to in poset theory are the linear extensions of a poset.

Definition 8. *Given a poset $P = (A, \succeq)$ with $|A| = N$. A* linear extension *of P is a function $f : P \to [N]$ such that f is bijective and*

$$x \succeq y \Rightarrow f(x) \geq f(y),$$

with $\geq$ the regular order relation on $\mathbb{N}$.

Our aim is to count the number of linear extensions of a specific poset, the so-called $n \times n$ lattice.

Figure 3.13: *The two possible linear extensions of the* 2×2 *lattice corresponding to the respective* $(2, 2)$ *Young Tableaux.*

Definition 9. *The poset* $(X_n, \gg)$ *with* $X_n = \{(i, j) \mid 1 \leq i \leq j \leq n\}$ *and* $(i, j) \gg (k, \ell)$ *if and only if* $i \geq k$ *and* $j \geq \ell$ *is called the* $n \times n$ lattice.

The following corollary now relates the number of linear extensions of the $n \times n$ lattice to the number of standard Young Tableaux of the shape $(n, \ldots, n)$.

Corollary 2. *The number of linear extensions of the* $n \times n$ *lattice is exactly* $f^{(n,\ldots,n)}$ *as given in Equation (3.2).*

Proof. Given a linear extension f of the $n \times n$ lattice, it induces an $(n, \ldots, n)$ Young Tableau by $(A_{ij})_{i,j=1,\ldots,n} = (f(i, j))_{i,j=1,\ldots,n}$. Furthermore, the map $f \mapsto (f(i, j))_{i,j=1,\ldots,n}$ is injective. □

This relationship gives another possible leverage on Conjecture 1. See Figure 3.13 for the two possible linear extensions of the 2×2 lattice and the two corresponding Young Tableaux of shape $(2, 2)$.

3.5 General Case, Parallelization, and Different Sorting Algorithms

3.5.1 General Case

In Section 3.1.2, we made several restrictions. In the following, we will show that the data structure—as well as the presented results and algorithms—do not suffer from these restrictions.

Not a square number of points A first assumption was that we are given exactly $N = n^2$ points in the point set P. How can the structure handle a general number of points? Given $m \in \mathbb{N}$ points $P = \{p_1, \ldots, p_m\}$ with $n^2 \geq m$ being the smallest square number larger or equal to m. Let $x_{\max}$ be the largest first coordinate and let $y_{\max}$ be the largest second coordinate, i.e.

$$x_{\max} := \max\{p_1^1, \ldots, p_m^1\}, \qquad y_{\max} := \max\{p_1^2, \ldots, p_m^2\}.$$

Add $n^2 - m$ auxiliary points of the form $(x_{\max} + 1, y_{\max} + 1)$ to P. In the algorithm of Theorem 3, these auxiliary points can be forced into the last two sequences Q_{n-1}, Q_n and thus be placed at the topmost rows or rightmost columns of the grid where they can be neglected in any neighborhood queries.

Larger dimension than $d = 2$ The second assumption was that the points $p_i \in P$ are two-dimensional. For higher dimensions, $d > 2$, consider a d-dimensional grid with side length n instead of a matrix, i.e. $M_\pi(P) \in (\mathbb{R}^d)^{n \times \ldots \times n}$. Denote by

$$p_i = (p_i^1, \ldots, p_i^d) \in P, \qquad p_j = (p_j^1, \ldots, p_j^d) \in P, \qquad p_i \neq p_j$$

two points stored at cells $(c_1, \ldots, c_d), (c'_1, \ldots, c'_d) \in [n]^d$ in $M_\pi(P)$, $(c_1, \ldots, c_d) \neq (c'_1, \ldots, c'_d)$. There are now d sorting conditions, as for each $\ell \in [d]$ there is one condition: Either $c_\ell \neq c'_\ell$ or w.l.o.g. $c_\ell < c'_\ell$ and

$$\begin{aligned}
&p_{i\ell} < p_{j\ell} \\
\vee &(p_{i\ell} = p_{j\ell} \wedge p_{i(\ell+1)} < p_{j(\ell+1)}) \\
\vee &\ldots \\
\vee &(p_{i\ell} = p_{j\ell} \wedge \ldots \wedge p_{i(d-1)} = p_{j(d-1)} \wedge p_{id} < p_{jd}) \\
\vee &(p_{i\ell} = p_{j\ell} \wedge \ldots \wedge p_{id} = p_{jd} \wedge p_{i1} < p_{j1}) \\
\vee &\ldots \\
\vee &(p_{i\ell} = p_{j\ell} \wedge \ldots \wedge p_{id} = p_{jd} \wedge p_{i1} = p_{j1} \wedge \ldots \wedge p_{i(\ell-2)} = p_{j(\ell-2)} \\
&\wedge p_{i(\ell-1)} < p_{j(\ell-1)}).
\end{aligned} \tag{3.3}$$

That is, in the ℓth dimension of the grid, two points are compared starting from the ℓth coordinate. The comparison is performed cyclically through all coordinates. The point p_i is considered to be smaller than the point p_j in the ℓth coordinate if starting from ℓ, the first coordinate where p_i and p_j differ is smaller in p_i.

Equal points If there are points $p_i, p_j \in P$ such that $p_i = p_j$, no total ordering can be imposed on the points using Equation (3.3). This tie can (in case of equality of the points) be easily broken by declaring p_i to be "smaller" than p_j if and only if $i < j$, cf. Footnote 7, page 21.

3.5.2 Iterative Parallelized Procedure

A benefit of the neighborhood grid data structure not discussed so far is the straight forward parallelization of an algorithm creating a stable state. In this section, we discuss the parallelization of a building algorithm for stable states, investigate its runtime, and elaborate on how iterative procedures can speed up the re-building of a stable state in case of addition or deletion of points.

The idea of iterative and parallel creation of stable states is discussed in [MW15] at great detail and we are only going to state the basic ideas and results here. However, some questions are not investigated in their paper, which we are going to tackle here. Parallelization of the algorithm from Theorem 3 will be discussed in Section 3.5.3. The general scheme for a two-dimensional grid is quite simple and given in Algorithm 3.

Algorithm 3 Iterative Parallelized Sorting of a Neighborhood Grid

```
1: procedure ITERATIVE PARALLELIZED SORTING(Grid G)
2:     while ¬sorted(G) do
3:         sort all rows of G in parallel
4:         sort all columns of G in parallel
5:     end while
6: end procedure
```

This algorithm can be completely run in parallel if $\sqrt[d]{N}$ processors are available, which should be realistic given the possibility to use GPU shaders for this task. An immediate question following this algorithm concerns its termination and correctness. Does it terminate and if so, is the final state stable? Malheiros and Walter did not answer this question in [MW15], nor did Joselli et al. in the original publications [Jos+09; Jos+15], where the idea of Algorithm 3 is also used. We will prove the termination here.

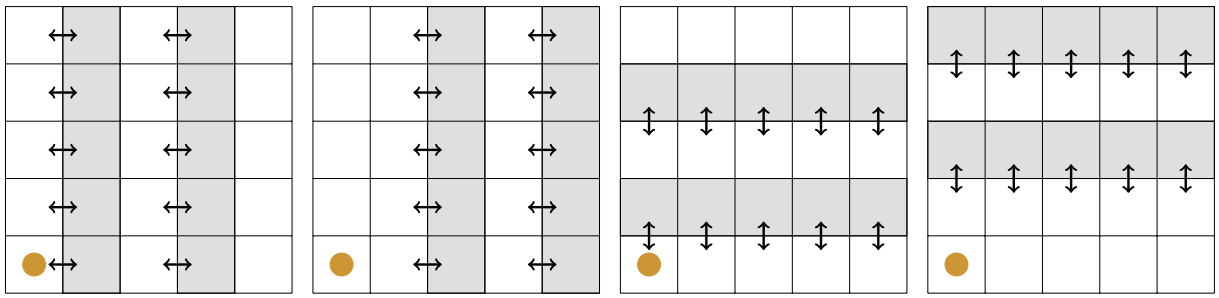


Figure 3.14: *Step-based odd-even sort on a two-dimensional neighborhood grid.*

Theorem 10 (Convergence of Algorithm 3, U. Reitebuch and M. S.)**.** *Given any point set $P \subset \mathbb{R}^2$ with N points, placing the points in a matrix $M_\pi(P)$ as described in Section 3.1.2, and running Algorithm 3 on the matrix, the algorithm terminates and yields a stable state of $M_\pi(P)$.*

Proof. Given a matrix $M_\pi(P)$ as in (3.1), consider the following expression:

$$E(M_\pi(P)) = \sum_{i,j=1}^{n} i \cdot p^1_{\pi^{-1}(i,j)} + j \cdot p^2_{\pi^{-1}(i,j)}, \tag{3.4}$$

for each sorting step of Algorithm 3, this expression grows strictly monotonically, but it can at most attain $N!$ many different values.

If the matrix is not in a stable state, i.e. there is a row or column violating the stable state, sorting this row or column lets expression (3.4) grow and resolves the conflict in the given row or column, possibly creating a new conflict in another row or column. Therefore, a local maximum of this expression is equivalent to a stable state in the matrix. □

Note that the energy functional (3.4) can easily be extended to higher-dimensional settings. Given a corresponding version of Algorithm 3 for the higher-dimensional case, Theorem 10 holds true for arbitrary dimensional point sets.

In their work [MW15], Malheiros and Walter investigate a slight variation of Algorithm 3. Namely, they do not perform a full sorting of a row or column, but rather consider one step of the odd-even sort algorithm of Habermann, see [Hab72]. Such step performs an exchange between all those cells in odd columns and their respective right neighboring cells, if this pair of cells violates the stable state conditions of Definition 3. Then, all cells in even columns and their respective right neighboring cells are compared and exchanged if necessary. The same is performed on all odd rows and even rows, yielding a four step mechanism, see Figure 3.14. The procedure can be run fully in parallel, if $\sqrt[d]{N} \cdot \lfloor \frac{\sqrt[d]{N}}{2} \rfloor$ processors are available.

Note that the argument in the proof of Theorem 10 holds true also for this algorithm. Therefore, it also terminates with a stable state. The question remains how fast the algorithm works asymptotically. Note that, defying intuition, elements can cycle through the matrix when using this procedure. An example is given in Figure 3.15, where the algorithm depicted in Figure 3.14 is used. Given the possibility of cycling elements, the theoretical asymptotic bounds of the parallel iterative algorithm remain unclear and only experimental results are available, as presented in [MW15].

Open Question 2. *Given the odd-even algorithm depicted in Figure 3.14, what is its parallelized worst-case runtime aside from the upper bound of $\mathcal{O}(N!)$ as established in the proof of Theorem 10?*

Our experiments let us to state the following conjecture:

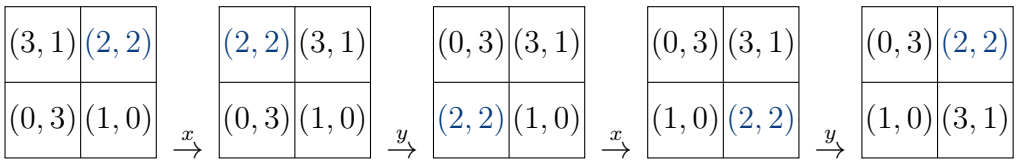


Figure 3.15: *Performing an odd-even sort alternating on all the rows and columns causes the blue element* $(2, 2)$ *to cycle through the matrix.*

Conjecture 2 (Worst-case runtime of stepwise odd-even sort, U. Reitebuch and M. Skrodzki)**.** *The step-wise odd-even algorithm depicted in Figure 3.14 runs like Bubble-Sort in worst case time of* $\mathcal{O}(N^2)$*, after parallelization in* $\mathcal{O}(N)$*.*

Compared to the single-core algorithm of Theorem 3, this would amount for a speed-up of only $\log(n)$ despite the usage of $\mathcal{O}(n)$ processors. Thus, in the following, we consider parallelization of the algorithm from Theorem 3 and compare the neighborhood grid to an exact neighborhood data structure.

3.5.3 Adaptive Sorting Algorithms for Fast Modifications, Comparison to k-d Trees, and Parallelization of Theorem 3

The most wide-spread data structure for neighborhood computation, k-d trees (see Section 2.1), popular for its expected nearest neighbor lookup time of $\mathcal{O}(\log(n))$ (see Theorem 2), suffers from a severe problem. Namely, if the underlying point set is slightly altered, the k-d tree might become unbalanced. Although there are some heuristics how to modify k-d trees when adding or deleting points, at some stage the k-d tree has to be rebuild, which is costly. The authors of [GG99] conclude to this end:

> *The adaptive k-d tree is a rather static structure; it is obviously difficult to keep the tree balanced in the presence of frequent insertions and deletions.*

In contrast, the neighborhood grid is built only with sorting algorithms. For these, adaptive algorithms are available that benefit from a sorted set into which a small number of records is to be inserted, cf. [PM92]. Thus, altering the point set underlying the neighborhood grid can be performed faster than rebuilding a k-d tree. An exact investigation of this relation is left for future research.

Note that the algorithm presented in Theorem 3 needs to sort the given points. When utilizing $N/2$ processors, sorting can be performed in $\log(N)$ time, see [AKS83]. Therefore, the presented algorithm can be parallelized to run in $\mathcal{O}(\log(N))$. This particular approach is of rather theoretical relevance, as the constants in [AKS83] are comparably large. However, other authors are devoted on finding more practical parallelizations, see [Ama+96]. Also, it makes for a significant speed-up compared to the algorithm depicted in Figure 3.14.

Compare this to building a k-d tree in parallel. A straight-forward parallelization would be as follows: In each step i, we have to sort i sets of $n^2/2^i$ points in the dimension with largest spread, which takes $\log(n^2) - i$ time for each of the $\log(n^2)$ levels of the tree, resulting in an upper bound for the total building time of $\Omega(\log^2(n))$. However, this only holds for a straight forward parallelization. As a k-d tree can be used for sorting by placing all points along one dimension, by [Lei85], it has a minimum build time of $\mathcal{O}(\log(n))$. To the best of our knowledge, it is unclear whether the gap between this lower bound and the upper bound induced by the straight forward parallelization can be closed.

Therefore, the neighborhood grid can be built slightly faster when compared with the straight forward parallelized k-d tree, but only gives estimated answers, while the k-d tree provides exact neighbor relations.

3.6 Quality of Neighborhood Approximation

As stated above, the neighborhood estimates given by the neighborhood grid data structure are not necessarily precise. In this section, we will investigate the quality of the neighborhood approximation.

3.6.1 Single Point Neighbor

A first question to answer in this section concerns the distance of two geometrical nearest neighbors from P in the stable state of $M_\pi(P)$. In the following, we will present a point set P with points $p, q \in P$ that are respective nearest neighbors to each other within P, but that lie on the exact opposite sites of $M_\pi(P)$.

Consider the following points $p = (0,0)$, $q = (1,1)$, $p_i = (0, 2 + i/n)$, $i = 1, \ldots, n-1$, $q_i = (1, -2 - i/n)$, $i = 1, \ldots, n-1$, and $r_{i,j} = (1 - 1/i, 2 + j/n)$, $i = 2, \ldots, n-1$, $j = 1, \ldots, n$. This yields a point set P with n^2 points for $n \geq 2$, see Figure 3.16a. Given these points, the following matrix is in a stable state:

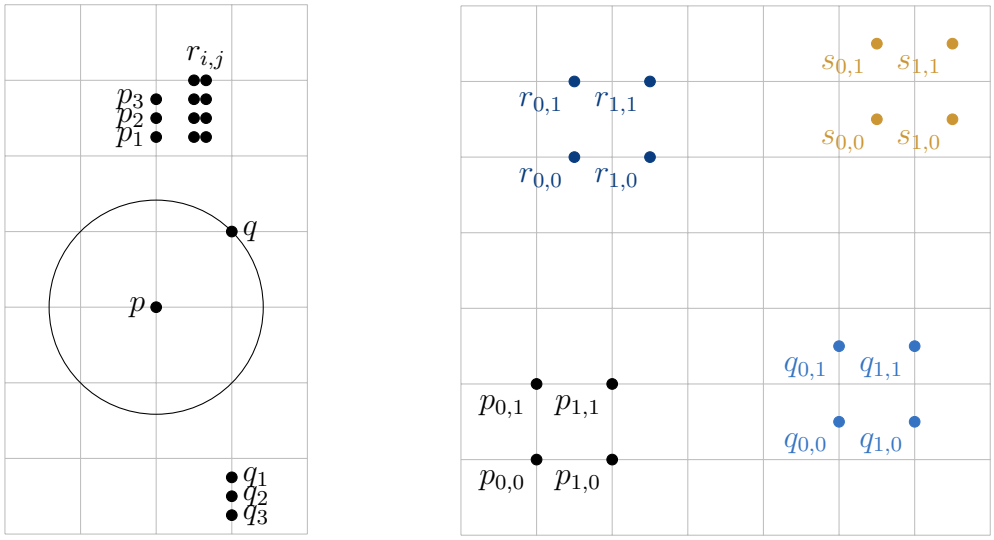
$$M_\pi(P) = \begin{array}{|c|c|c|c|c|} \hline p_{n-1} & r_{2,n} & \cdots & r_{n-1,n} & q \\ \hline \vdots & \vdots & \iddots & \vdots & q_1 \\ \hline p_1 & r_{2,2} & \cdots & r_{n-1,2} & \vdots \\ \hline p & r_{2,1} & \cdots & r_{n-1,n} & q_{n-1} \\ \hline \end{array}.$$

Note that the nearest neighbor to p and q in P is q and p respectively. However, in $M_\pi(P)$, these points lie in the opposing corners of the matrix. That is, in order to find the geometrically closest neighbor to p in $M_\pi(P)$, the n-ring around p has to be checked. In other words, all points have to be checked, which takes $\Theta(n^2)$ compared to the expected time of $\mathcal{O}(\log(n))$ as in k-d trees (see Theorem 2).

3.6.2 All Point Nearest Neighbors

In the previous example, we saw that for a single point, its unique nearest neighbor can be arbitrarily far away in the neighborhood grid. However, when considering all points, how is the overall estimate? Here, we provide an example, where no point has its corresponding neighbor within its one-ring in the neighborhood grid.

For $n \in \mathbb{N}$, $n \mod 2 \equiv 0$, consider the following points $p_{i,j} = (i, j)$, $q_{i,j} = (i + n, j + 0.5)$, $r_{i,j} = (i + 0.5, j + n)$, and $s_{i,j} = (i + n + 0.5, j + n + 0.5)$ for $i, j \in \{0, \ldots, \frac{n}{2} - 1\}$. This yields a point set P with n^2 points, see Figure 3.16b. Given these points, the following matrix is in a



(a) *Point set P as given in Section 3.6.1.* **(b)** *Point set P as given in Section 3.6.2 for $n = 4$.*

Figure 3.16: *Illustration of extremal examples where two geometrical nearest neighbors lie arbitrary far from each other in the neighborhood grid and where no point of the input set P has its geometrical nearest neighbor within its one-ring in the neighborhood grid.*

stable state:

$$M_\pi(P) = \begin{array}{|c|c|c|c|c|c|c|} \hline q_{0,\frac{n}{2}-1} & s_{0,\frac{n}{2}-1} & q_{1,\frac{n}{2}-1} & s_{1,\frac{n}{2}-1} & \cdots & q_{\frac{n}{2}-1,\frac{n}{2}-1} & s_{\frac{n}{2}-1,\frac{n}{2}-1} \\ \hline p_{0,\frac{n}{2}-1} & r_{0,\frac{n}{2}-1} & p_{1,\frac{n}{2}-1} & r_{1,\frac{n}{2}-1} & \cdots & p_{\frac{n}{2}-1,\frac{n}{2}-1} & r_{\frac{n}{2}-1,\frac{n}{2}-1} \\ \hline \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \hline q_{0,1} & s_{0,1} & q_{1,1} & s_{1,1} & \cdots & q_{\frac{n}{2}-1,1} & s_{\frac{n}{2}-1,1} \\ \hline p_{0,1} & r_{0,1} & p_{1,1} & r_{1,1} & \cdots & p_{\frac{n}{2}-1,1} & r_{\frac{n}{2}-1,1} \\ \hline q_{0,0} & s_{0,0} & q_{1,0} & s_{1,0} & \cdots & q_{\frac{n}{2}-1,0} & s_{\frac{n}{2}-1,0} \\ \hline p_{0,0} & r_{0,0} & p_{1,0} & r_{1,0} & \cdots & p_{\frac{n}{2}-1,0} & r_{\frac{n}{2}-1,0} \\ \hline \end{array}.$$

Note that the nearest neighbor to $p_{i,j}$ is some $p_{k,\ell}$, the nearest neighbor to $q_{i,j}$ is some $q_{k,\ell}$, and so on. However, none of the points has its corresponding neighbor in its one-ring. Expanding this scheme by adding more four-point subsets, it is easily achievable to create point sets with immediate stable states where no respective nearest neighbor is in the $\frac{n}{4}$ ring of all points and so on.

3.6.3 Experimental Results

We have given two constructions to create worst case behavior in the neighborhood estimates. However, these constructions do not account for the average estimate to be expected for a random point set. Therefore, we pose the following questions:

Open Question 3. *How good is the average neighborhood estimate of the neighborhood grid? How does the building procedure affect the neighborhood quality?*

Concerning both questions, we present experimental results here. We generated 70 point sets with $1,764$ points each randomly in $[0,1]^2 \subset \mathbb{R}^2$. They are randomly placed into the grid. Then, each of the following four methods is applied in order to build a stable state on the grid:

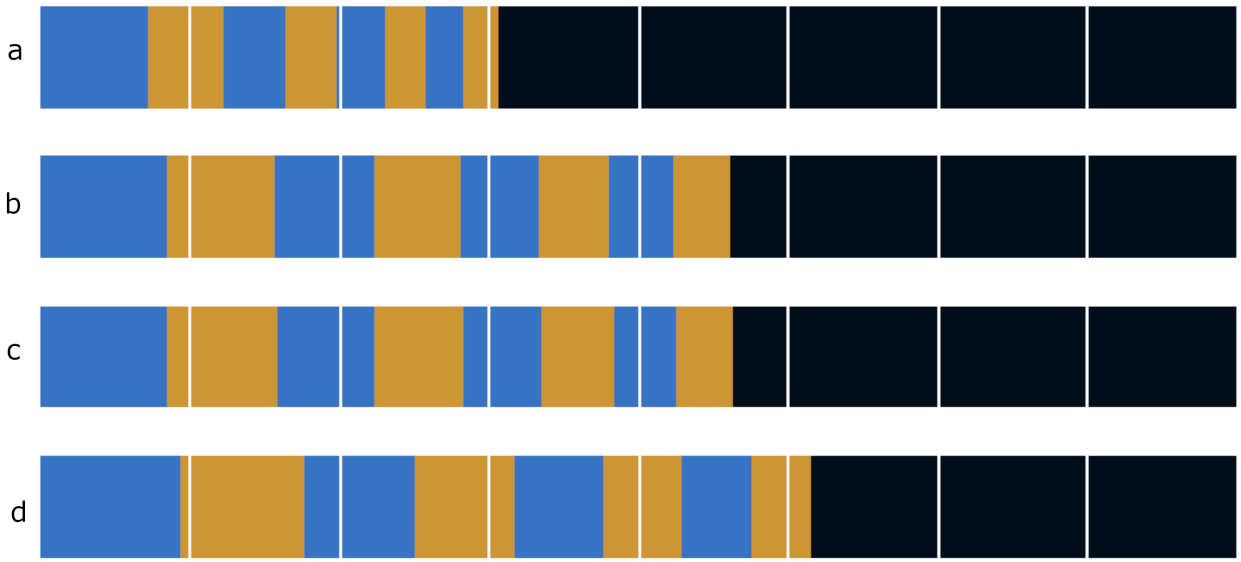


Figure 3.17: *From top to bottom: Percentage of neighbors correctly reporting their 1st, 2nd, . . ., 8th nearest neighbors when using (a) direct sorting following Theorem 3, (b) performing complete odd-even-sort on all rows and columns alternating, (c) performing one step odd-even-sort on all rows and columns alternating, (d) exchanging the pair of points which provide largest grow in Energy (3.4). From left to right, the boxes indicate whether every point knows its nearest, its two nearest, . . ., its eight nearest neighbors amongst its one-ring of eight points. The colored bars indicate to what percentage the respective boxes are filled. Note that the direct method of Theorem 3 runs fastest among these methods but provides the worst results. Exchanging two points for maximum grow of Energy (3.4) is the most time-consuming procedure but provides the best results.*

(a) The grid is sorted directly following the algorithm from Theorem 3.

(b) All rows are sorted, then all columns are sorted. The procedure is iterated until convergence.

(c) The step-based odd-even sort as shown in Figure 3.14 is performed.

(d) We exchange those points p_i, p_j in the grid such that the grow in Energy 3.4 is maximal.

As the output of procedures b–d heavily depends on the initial placement of the points in the grid, we apply the procedure on 30 different random placements of each point set. Finally, we count how many points in the grid have their nearest neighbor, their two nearest neighbors, . . ., and their eight nearest neighbors within their one-ring. The results are shown in Figure 3.17. Note that the most time-consuming methods—exchanging by maximum growth of Energy (3.4)—gives the best results while the direct sorting following Theorem 3 performs worst despite being the fastest sorting method.

3.7 Conclusion and Future Work

In this section we have presented the neighborhood grid data structure of Joselli et al. [Jos+09; Jos+15]. Starting from the results of Malheiros and Walter [MW15], we were able to prove asymptotic time-optimality of a deterministic building algorithm in Theorem 5.

The proof required the investigation of several combinatorial questions of which some were answered in Theorem 4 as well as Section 3.3.3. However, some combinatorial questions remain

open. Namely, it is unclear what the least number of stable states for a given grid size n is, see Open Question 1. For the dual question, asking for the largest number of stable states for a given grid size n, we presented Conjecture 1.

Concerning the run time of the parallelized algorithm, the exact asymptotic time remains unclear, see Open Question 2, for which we also present a conjecture, see Conjecture 2. Finally, the quality of the neighborhoods given by the data structure was investigated experimentally. Thorough proofs for neighborhood qualities are left—like the open questions and conjectures—as future work.

II Manifold Structure for Point Set Surfaces

4 Manifold Theory and Formulations for Point Set Manifolds

In practical applications, digital point sets are representations of real-world objects. Although there are acquisition techniques like MRI which provide a volumetric representation of a 3D real-world object, all vision-based processes—like 3D (laser) scanning—only provide information about the surface of the object. The fundamental question we will investigate in this section concerns the search for a convenient description of a point set surface in terms of a manifold-like chart structure.

Given $d, d' \in \mathbb{N}_0$, $d' \leq d$, a d'-manifold $\mathcal{M} \subseteq \mathbb{R}^d$, $\mathcal{M} = \bigcup_i U_i$, U_i open, comes with mappings $\varphi_i : U_i \to \mathbb{R}^{d'}$, with i in some index set I (refer to Definitions 10 and 11 for a precise formulation). In a discrete setting acting with point sets, these U_i have to be modeled by neighborhoods as no other structures can be imposed on a point set. A prominent first choice consists in modifications of a k nearest neighborhood, see (1.1). The fast generation of such neighborhoods with different data structures has been discussed in Sections 2 and 3. However, in general, a combinatorial k nearest neighborhood is not a good notion for several aspects, such as curvature or normal estimates. The reason is that this type of neighborhood does not take into account the different densities that arise in a point set during acquisition (see Section 6 for a more thorough discussion). Hence, an ε neighborhood—see (1.4)—is often a better choice and is just as effective, see [Skr14b]. However, it does not explain the generation of charts or transition maps between these. The major task in this section is therefore to find a meaningful description of manifold-like structures for point sets.

On a slightly more global perspective, geometric point-clusters provide a coarse segmentation of the geometry into parts that are formed by points which are close to each other in a given similarity measure[1]. Again, it is an active topic in computational geometry how to analyze clustering algorithms in terms of complexity and statistical properties, see [FHT01; Bis06]. This is in particular true because of the connections to the vast topic of big data analysis, see [Fah+14]. However, these clusters do not overlap and thus do not mimic open sets on a manifold.

Yet another approach is to not compute exact neighborhoods, but only approximate them in favor of better runtime behavior. This approach can be realized using provably efficient data structures, as discussed on the example of the neighborhood grid in Section 3. A different approach to neighborhood approximation is taken for example by the authors of [Mul+14] who present a fairly elaborate data structure which can answer an approximate k-flat nearest neighbor query

[1] This measure could e.g. be a bilateral measure, incorporating both normal information and euclidean distance, cf. the procedure of Section 1.

on n points in time $\mathcal{O}(n^{k/(k+1-\rho)+t})$ for some $\rho, t > 0$.

We would like to take the—from a differential geometric perspective—next step and define a manifold-like structure on a point set representing a d'-manifold. Thus, the two main contributions of this section are:

- Establishing that $d' = 0$ is an impractical choice for the manifold dimension of a point set.
- Definition of a scheme to treat point sets as manifolds via a transition manifold.

4.1 Definition of a Smooth Manifold

In this section, we review the definition of a manifold. A thorough account on smooth manifolds is [Lee12]. The terminology set here will be used in the following sections.

Definition 10. *A topological space $(\mathcal{M}, \mathcal{T})$ of a set $\mathcal{M}$ and a topology $\mathcal{T}$ on $\mathcal{M}$ is called a* topological d'-manifold, *$d' \in \mathbb{N}_0$, if the following requirements are met.*

1. *$(\mathcal{M}, \mathcal{T})$ is a* Hausdorff space. *That is, for all elements $p_i, p_j \in \mathcal{M}$ there are open subsets $U_i, U_j \in \mathcal{T}$ such that $U_i \cap U_j = \emptyset$, $p_i \in U_i$, and $p_j \in U_j$.*
2. *$(\mathcal{M}, \mathcal{T})$ is* second countable. *That is, there exists a countable basis for $\mathcal{T}$.*
3. *$(\mathcal{M}, \mathcal{T})$ is* locally Euclidean *of dimension d'. That is, for every element $p_i \in \mathcal{M}$, there exist open subsets $U_i \subset \mathcal{M}$, $U_i' \subset \mathbb{R}^{d'}$ with $p_i \in U_i$ and a homeomorphism $\varphi : U_i \to U_i'$.*

We will now further elaborate on the local descriptions of the manifold as part of $\mathbb{R}^{d'}$.

Definition 11. *Let $(\mathcal{M}, \mathcal{T})$ be a topological d'-manifold.*

- *A* (coordinate) chart *on $(\mathcal{M}, \mathcal{T})$ is a pair (U_i, φ), where $U_i \in \mathcal{T}$ and $\varphi : U_i \to U_i' \subset \mathbb{R}^{d'}$ a homeomorphism with U_i' open in $\mathbb{R}^{d'}$.*
- *If (U_i, φ_i), (U_j, φ_j) are two charts with $U_i \cap U_j \neq \emptyset$, then the map $\varphi_j \circ \varphi_i^{-1} : \varphi_i(U_i \cap U_j) \to \varphi_j(U_i \cap U_j)$ is called the* transition map *from φ_i to φ_j.*
- *A set $\mathcal{A}$ of charts is called an* atlas *of $(\mathcal{M}, \mathcal{T})$ if the domains of the charts cover $\mathcal{M}$.*

Finally, we want to impose the notion of smoothness on the manifold. Having atlases and transition maps at hand, we can do this with the following definition.

Definition 12. *Let $(\mathcal{M}, \mathcal{T})$ be a topological d'-manifold with at least two charts (U_i, φ_i), (U_j, φ_j). The charts are called* smoothly compatible *if either $U_i \cap U_j = \emptyset$ or the transition map $\varphi_j \circ \varphi_i^{-1}$ is a diffeomorphism, i.e. both $\varphi_j \circ \varphi_i^{-1}$ and its inverse are in $\mathcal{C}^\infty$.*
An atlas $\mathcal{A}$ is called smooth, *if all charts in $\mathcal{A}$ are pairwise smoothly compatible. A smooth atlas $\mathcal{A}$ on $\mathcal{M}$ is* maximal *if it is not contained in any smooth atlas with strictly more charts.*
A smooth d'-manifold *is a pair $((\mathcal{M}, \mathcal{T}), \mathcal{A})$ of a topological d'-manifold $(\mathcal{M}, \mathcal{T})$ and a maximal smooth atlas $\mathcal{A}$.*

4.2 Point Sets as 0-Manifolds

A first intuitive approach is to consider a given point set $P = \{p_1, \dots, p_n\}$ as zero-dimensional manifold, i.e. choosing $d' = 0$. This is to say that the set $\mathcal{M} = P$ is given the topology $\mathcal{T}$ generated by the open sets $U_i = \{p_i\}$, $i \in [n]$. Given this basis, $(\mathcal{M}, \mathcal{T})$ is Hausdorff, since $p_i, p_j \in \mathcal{M}$ can be separated by the open sets $U_i, U_j \in \mathcal{T}$. Furthermore, $(\mathcal{M}, \mathcal{T})$ is second countable since the basis $\{U_i \mid i \in [n]\}$ for $\mathcal{T}$ is finite. Finally, the manifold $(\mathcal{M}, \mathcal{T})$ is also locally Euclidean of dimension

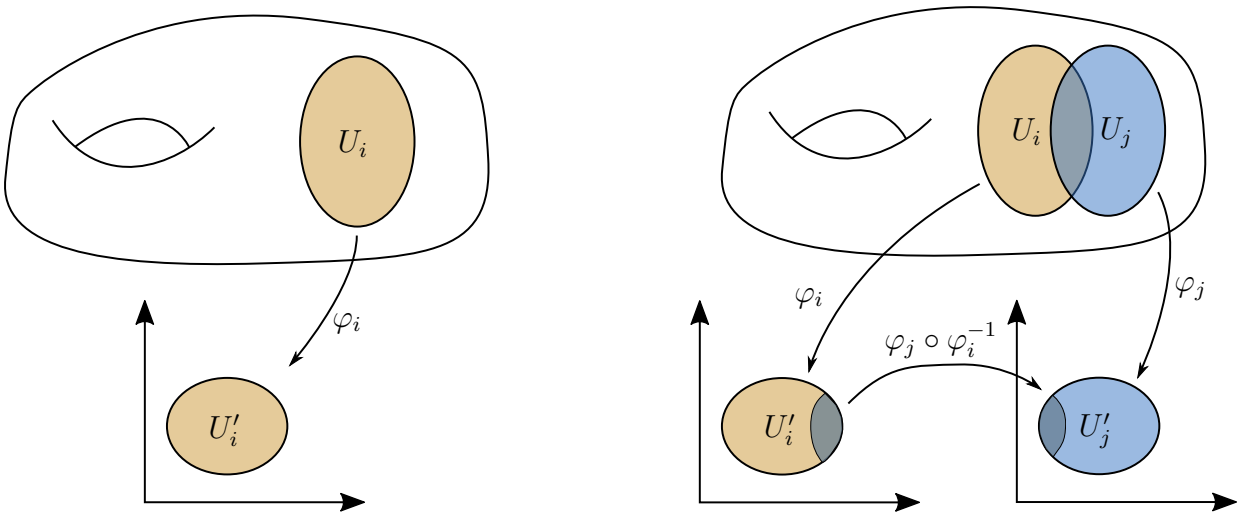


Figure 4.1: *An illustration of a chart on the left and a transition map on the right as given in Definition 11.*

$d' = 0$, since for each $p_i \in M$, the map $\varphi : U_i = \{p_i\} \to \mathbb{R}^0 = \{0\}$ is a homeomorphism. Thus, $\mathcal{M}$ satisfies all properties of a topological manifold.

With the construction as given above, $(\mathcal{M}, \mathcal{T})$ is even a smooth 0-manifold as specified in Definition 12. The only possible 0-dimensional subsets of $\mathcal{M}$ to be mapped to $\mathbb{R}^0 = \{0\}$ under a homeomorphism are those consisting of a single element. Thus, the $U_i = \{p_i\}$ as given above are the only possible open subsets and hence the atlas $\mathcal{A} = \{\varphi_i : U_i \to \mathbb{R}^0 \mid i \in [n]\}$ is maximal. As $U_i \cap U_j = \emptyset$ for all $i, j \in [n]$, $i \neq j$, the maps φ_i are smoothly compatible.

However, regarding the point set P as a 0-manifold $\mathcal{M}$ leads to $\mathcal{M}$ being disconnected, as the following theorem shows.

Theorem 11 (Finite Hausdorff spaces are disconnected, Elementary Topology)**.** *Given a topological space $(\mathcal{M}, \mathcal{T})$ that is Hausdorff, consists of finitely many elements, and includes at least two elements. Then each set of the form $\{p_i\}$, $p_i \in \mathcal{M}$ is open and $(\mathcal{M}, \mathcal{T})$ is disconnected.*

Proof. Let $p_i \in \mathcal{M}$ be some arbitrary element. Since $(\mathcal{M}, \mathcal{T})$ is Hausdorff, for every $p_j \in \mathcal{M}$, there exists some $U_{\bar{j}} \in \mathcal{T}$ such that $p_i \in U_{\bar{j}}$ and $p_j \notin U_{\bar{j}}$. Then the intersection

$$\bigcap_{p_j \in \mathcal{M} \setminus \{p_i\}} U_{\bar{j}} = \{p_i\}$$

is an element of $\mathcal{T}$ and thus open. Furthermore, $\bigcup_{p_j \in \mathcal{M} \setminus \{p_i\}} \{p_j\}$ is open, so is $\{p_i\}$ and their union gives $\mathcal{M}$. Hence, $(\mathcal{M}, \mathcal{T})$ is disconnected. □

This simple theorem shows that the discussion above is somewhat independent of the topology imposed on the point set P, since the demand of the topological space being Hausdorff already implies P to be disconnected, when considered as a manifold.

The disconnectedness of the 0-manifold $\mathcal{M}$ disqualifies 0-manifolds as practical objects in the context of geometry processing. Thus, a different formulation has to be found for finite point sets to represent a manifold.

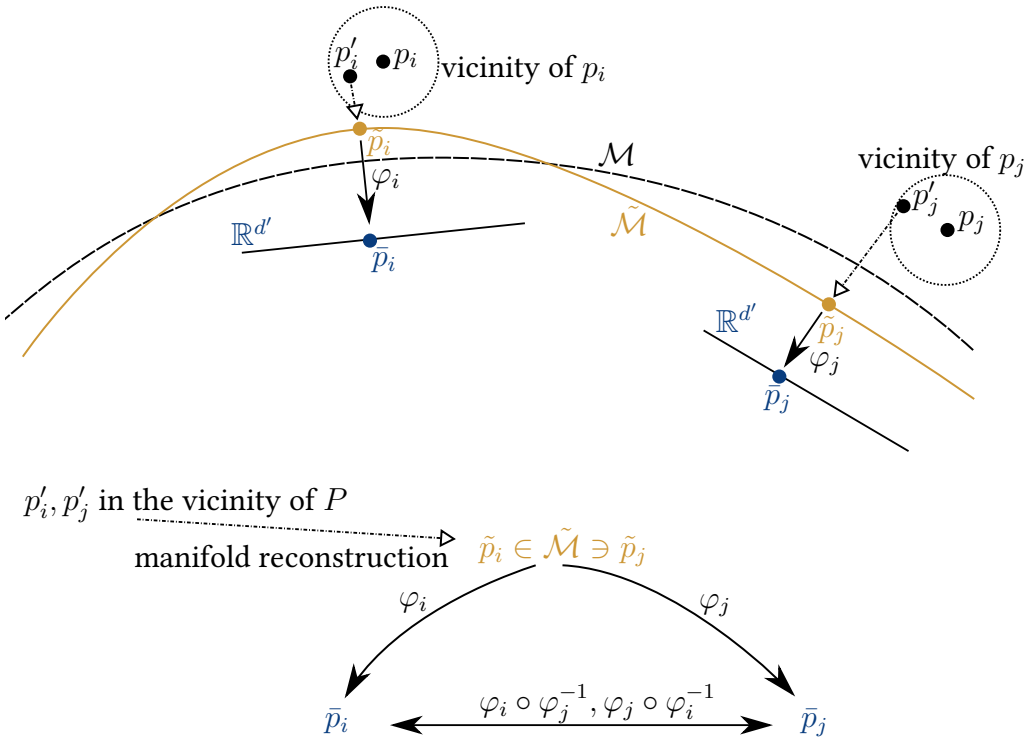


Figure 4.2: *An illustration of the procedure to obtain smooth transition maps from a point set P by using manifold reconstruction and the charts φ, ψ of the reconstructed manifold $\tilde{M}$.*

4.3 Recovered d'-Manifolds

We have seen above that the simple approach of perceiving a point set as 0-dimensional manifold does not lead far. Therefore, consider the following setup to remedy this problem. In order to increase readability, in the following, we will refer to a manifold $(\mathcal{M}, \mathcal{T})$ simply by $\mathcal{M}$.

Suppose there exists a smooth d'-manifold $\mathcal{M}$ from which only a possibly noisy sample $P \subset \mathbb{R}^d$ of n points $p_i \in P$ is known. Consider a smooth d'-manifold $\tilde{\mathcal{M}}$ as a reconstruction of $\mathcal{M}$ from P. We will discuss in Section 4.4 how this reconstruction can be obtained. Then, for points p_i', p_j' in the vicinity of $p_i, p_j \in P$ respectively, we can perform the following: Project p_i', p_j' onto $\tilde{\mathcal{M}}$ and obtain $\tilde{p}_i, \tilde{p}_j$ respectively. Since $\tilde{\mathcal{M}}$ is a manifold, there exist open subsets $U_i, U_j \subseteq \tilde{\mathcal{M}}$ with $\tilde{p}_i \in U_i$, $\tilde{p}_j \in U_j$ and smoothly compatible coordinate charts $\varphi_i : U_i \to \mathbb{R}^{d'}$, $\varphi_j : U_j \to \mathbb{R}^{d'}$ with $\varphi_i(p_i') = \bar{p}_i \in \mathbb{R}^{d'}$ and $\varphi_j(p_j') = \bar{p}_j \in \mathbb{R}^{d'}$. Thus, by projecting a small vicinity of p_i and p_j onto $\tilde{\mathcal{M}}$ and utilizing the transition map induced by φ_i and φ_j, one can realize a transition map acting outside of the reconstructed manifold $\tilde{\mathcal{M}}$ on the sampled points. Note that none of this requires the manifold $\mathcal{M}$ to be known or to be embedded in Euclidean space. Also, the point samples P do not necessarily have to be embedded in Euclidean space. The only requirement is that there is some similarity measure[2] on the space from which the p_i are taken and that a unique projection onto the reconstructed manifold $\tilde{M}$ is possible, i.e. the sample P cannot be too noisy or too sparse. This idea is depicted in Figure 4.2.

The remaining question is: What is a suitable procedure to compute the reconstructed manifold $\tilde{\mathcal{M}}$? In the following section, we will describe one possibility based on the MLS framework. A different approach is given in Section 5.

[2] See Footnote 1 on page 51.

4.4 Manifold Reconstruction using the Moving Least Squares Approach

The Moving Least Squares (MLS) approach goes back to 1976 when introduced by McLain [McL76] as a tool for function approximation. Ever since, it has been improved by many researchers and translated to the context of surface approximation. If translated straight forward, the resulting formulation for surfaces is not a projection procedure. In order to have projection properties for the pipeline, the key change was to not consider the least squares distances $\|p - p_i\|^2$ from a point $p \in \mathbb{R}^d$ to samples $p_i \in \mathbb{R}^d$, but rather to consider the least square distance $\|\mathbb{MLS}(p) - p_i\|^2$ of the yet unknown projection $\mathbb{MLS}(p)$ to the samples. These important aspects have been introduced by David Levin [Lev04] and were subsequently used in the context of surface computation and rendering [Ale+03]. Only quite recently, the MLS approach was extended to smooth manifolds embedded in Euclidean space of arbitrary co-dimension [SL16]. Furthermore, the approach can be used for the approximation of functions over manifolds [SAL17].

4.4.1 Introduction to Moving Least Squares

For an introduction to the Moving Least Squares (MLS) approach, we refer to [Nea04]. Here, we will present the main results of [SL16] to the degree necessary for this thesis.

Sober and Levin, [SL16], build on previous work, see [Lev98; Lev04]. Even though their results hold for smooth manifolds of arbitrary dimensions, in the following we will state all results in the less general setting of manifolds of co-dimension 1—i.e. $d' = (d-1)$—as this is sufficient for our purpose.

Assume there exists some smooth, unknown $(d-1)$-dimensional manifold $\mathcal{M}$ embedded into $\mathbb{R}^d$, sampled by a set of points $P = \{p_i \in \mathbb{R}^d \mid i \in [n]\}$. The aim is to find an implicit description of a $(d-1)$-dimensional manifold $\tilde{\mathcal{M}}$ approximating $\mathcal{M}$ using P. For any point $p \in \mathbb{R}^d$, this is done via projecting p onto $\tilde{\mathcal{M}}$, which is realized in two steps:

1. Find a local approximating hyperplane H_p to serve as a local coordinate system close to p.
2. Project p using a local MLS approximation of $\mathcal{M}$ over the coordinate system induced by H.

This procedure is possible since the surface can be viewed locally as a graph of a function. We will now elaborate on these two steps.

Step One of the MLS procedure Given $p \in \mathbb{R}^d$ and sample points $p_i \in P \subset \mathbb{R}^d$, we want to find a hyperplane H_p which locally best approximates the point samples from P around p. We define the hyperplane via its unit-length normal vector $a \in \mathbb{R}^d$ and its distance $s \in \mathbb{R}$ to p. In other words, the hyperplane H_p is defined by a and placed at $q = p + sa$, see Figure 4.3, resulting in

$$H_p = \{x \in \mathbb{R}^d \mid \langle a, x\rangle = \langle a, p + sa\rangle\}. \tag{4.1}$$

The normal direction a as well as the value s are found by minimizing the following energy functional

$$\begin{aligned} I_p(a,s) &= \sum_{i=1}^{n} \left(\langle a, p_i\rangle - \langle a, p + sa\rangle\right)^2 \cdot \theta(\|p_i - (p + sa)\|), \\ &\text{s.t. } \|a\| = 1. \end{aligned} \tag{4.2}$$

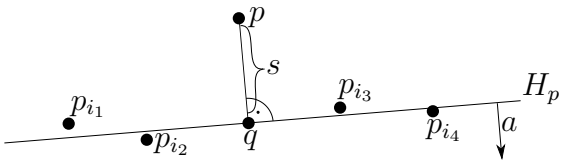


Figure 4.3: *Illustration of the local hyperplane H_p obtained by step one of the MLS procedure, see Section 4.4.1.*

Here, θ is a non-negative weight function to localize the influence of the p_i on H_p. Thus, we pick a $\theta \in C^\infty$ of local support of radius $r > 0$, namely

$$\theta(x) = \begin{cases} e^{-x^2/(x-r)^2} & \text{for } x \in (-r, r), \\ 0 & \text{otherwise} \end{cases}. \tag{4.3}$$

In any practical application, r has to be chosen according to the density of the point sample and the curvature of the underlying manifold. It can also be chosen differently for each point p. Note that the parameters a and s both appear inside the weight function θ, therefore minimizing expression (4.2) cannot be performed as in [Nea04] via a linear system, but is more involved. See [SL16] for an iterative procedure to find a pair of minimizers (a, s).

A first important result, following from (4.1) and (4.2) is the following theorem.

Theorem 12 (Projection Property of MLS, [SL16]). *Let $p \in \mathbb{R}^d$ lie sufficiently close to $\mathcal{M}$ and let (a, s) be a pair of minimizers of (4.2). Then, for any point $\tilde{p} \in \mathbb{R}^d$ also close enough to $\mathcal{M}$ with $\tilde{p} = p + \tilde{s}a$, $\tilde{s} \in \mathbb{R}$ we have*

$$\arg\min I_p(a, s) = \arg\min I_{\tilde{p}}(a, s - \tilde{s}).$$

In other words, any point $\tilde{p}$ on the line $c \cdot (p - q) + p$, $c \in \mathbb{R}$, that is still sufficiently close to $\mathcal{M}$ yields the same hyperplane $H_{\tilde{p}} = H_p$. For a formulation of the different bounds on distances and a proof of this theorem, we refer to [Lev04].

Step Two of the MLS procedure Given $p \in \mathbb{R}^d$, sample points $p_i \in P \subset \mathbb{R}^d$, $q = p + sa \in \mathbb{R}^d$ as obtained from step one described above. Denote by $q_i \in \mathbb{R}^{d-1}$ the orthogonal projection of p_i to H_p expressed in the local coordinate system on H_p and by

$$f_i = \langle p_i - q, a \rangle \in \mathbb{R}^1$$

the height of p_i over the local coordinate system defined by H_p, see Figure 4.4. Furthermore, consider $\prod_m^{d-1}$ to be the set of all $(d-1)$-variate polynomials of total degree m. We find a polynomial $\mathcal{P}_p$ as

$$\mathcal{P}_p = \arg\min_{\mathcal{P} \in \prod_m^{d-1}} \sum_{i=1}^{n} (\mathcal{P}(q_i) - f_i)^2 \cdot \theta(\|p_i - q\|). \tag{4.4}$$

In contrast to the minimization in (4.1), the weights $\theta(\|p_i - q\|)$ are now fixed and not a part of the minimization process. Therefore, Equation (4.2) can be solved directly by solving a linear system, see [Nea04]. For an illustration, see Figure 4.4. Now, the manifold $\mathcal{M}$ can be approximated at the point p by the approximating point p' given as

$$p' = q + \mathcal{P}_p(0) \cdot \frac{p - q}{\|p - q\|}. \tag{4.5}$$

The central theorem of the MLS approach can now be stated as follows:

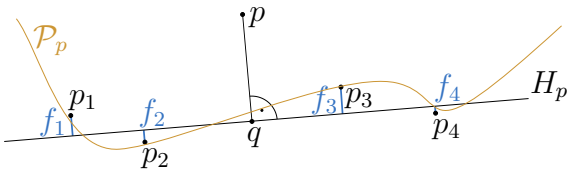


Figure 4.4: *Illustration of the local polynomial $\mathcal{P}_p$ obtained by Step Two of the MLS procedure, see Section 4.4.1.*

Theorem 13 (Approximation power of MLS, [SL16])**.** *Given a smooth manifold $\mathcal{M}$, assume that P is a suitable sample of $\mathcal{M}$ in the sense that Theorem 12 requires. Denote by f_q the distance of q to the intersection point of the line $\bar{pq}$ with $\mathcal{M}$. The order of the approximation error is then given as follows*

$$|\|p' - q\| - f_q| \in \mathcal{O}(h^{m+1}), \tag{4.6}$$

with p' the MLS projection of p given by Equation 4.5.

For detailed requirements imposed on the sample and a proof, we refer to [SL16]. In the following, we will refer to the whole projection procedure which maps p to p' by

$$\mathbb{MLS}_{P,r}(p) = p', \tag{4.7}$$

indicating that the sample set P and radius r was used in the procedure.

4.4.2 Moving Least Squares for Chart Construction

Given the MLS procedure as described above, the reconstruction of a manifold from a point sample P—as aimed for in Section 4.3—can be performed in the following way: Given a possibly noisy point sample P of an unknown smooth manifold $\mathcal{M}$. Consider the MLS approximation $\tilde{\mathcal{M}}$ of $\mathcal{M}$ obtained from P. Then for each point $p_i, p_j \in P$, during the MLS approximation we perform two steps. As described in 4.4.1, the first step yields local hyperplane approximations H_{p_i}, H_{p_j} for p_i, p_j respectively. These local hyperplanes will serve as two-dimensional coordinate charts providing $(d-1)$-dimensional coordinates for all points in P.

The second step of the MLS approximation—as described in Section 4.4.1—now defines polynomials $\mathcal{P}_{p_i}, \mathcal{P}_{p_j}$ on the respective local hyperplanes which in turn provide a height of the points of P over the respective hyperplanes. Therefore, the inverse maps of φ_i and φ_j—projections from $\tilde{\mathcal{M}}$ onto H_{p_i} and H_{p_j} respectively—simply consist of taking a $(d-1)$-dimensional point from H_{p_i} or H_{p_j} respectively and equipping it with the height from the corresponding polynomial. Thereby, one obtains a point in d-dimensional space: $\tilde{p}_i, \tilde{p}_j \in \tilde{M}$. The whole process is illustrated in the diagram of Figure 4.5, also compare with the more general Figure 4.2.

The coordinate charts φ_i, φ_j are projections from $\mathbb{R}^d$ to H_{p_i}, H_{p_j}, restricted to $\tilde{M}$. They are smooth and bijective by the assumptions of Theorem 12. Furthermore, their inverse map consists in the application of a polynomial which is smooth by definition. Thereby, we have obtained smooth coherent transition maps that mimic the behavior of smooth transition maps on a smooth manifold.

4.5 Conclusion: Local versus Global Charts

In the chart construction procedure given in Section 4.4.2, a chart is constructed for every element of the point set P. That is, if the geometry has locally flat areas, the procedure will still create

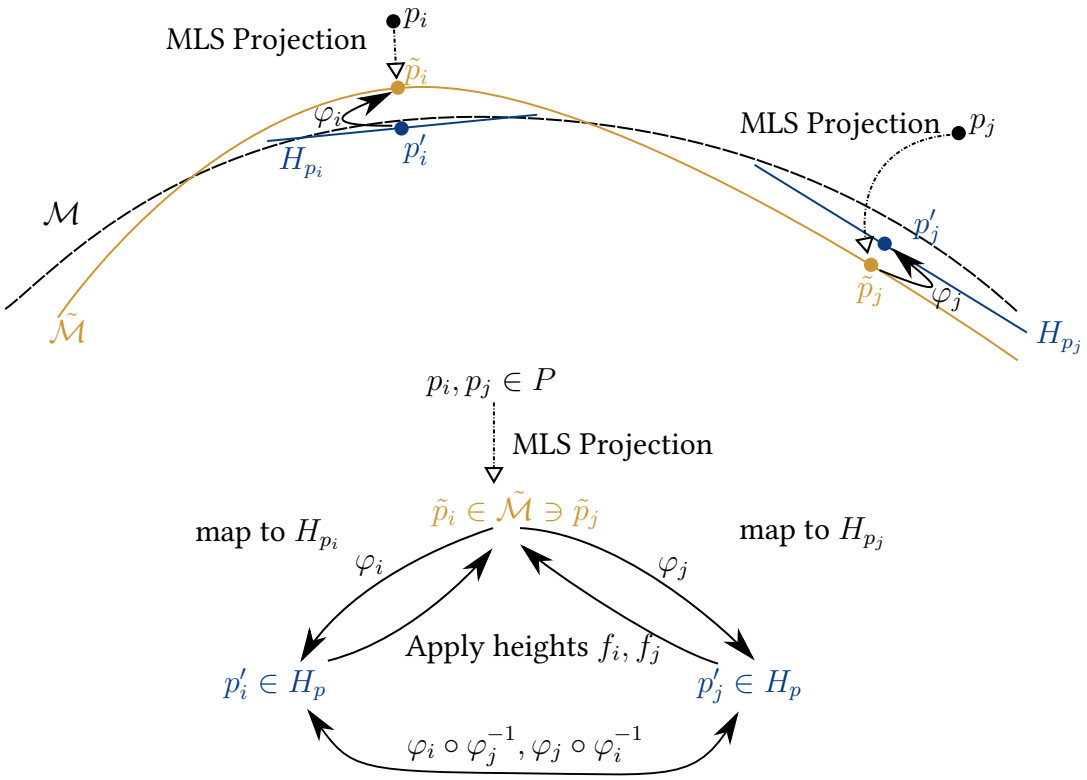


Figure 4.5: *An illustration of the procedure to obtain smooth transition maps from a point set P by using MLS.*

charts for each sample in these areas, even though the charts in one area will all be very similar up to the approximation order of Theorem 13. Thus, this procedure takes the role of creating charts that map a small region of the whole geometry. In this sense, these charts are very local. Note that the charts can even be created for points outside of the original sample set P, but will still be localized at a chosen base point.

A completely different approach is taken by Li et al. [Li+11]. The authors consider a concept originally introduced by Kälberer, Nieser, and Polthier [KNP07], which was reformulated by Bommes, Zimmer, and Kobbelt [BZK09]. This concept is translated to the setting of point sets. The authors first compute a direction field acting on all points of the sample set P. This direction field is then guiding a meshless global parametrization. For this, the point set is cut open along a graph such that it becomes a topological disc. On the disc, the point set is parametrized. Then, along the cut-graph, the gradients of the parametrization are fit such that possible error are distributed and the parametrization lines fit along the cut-graph. In order to obtain a quad-parametrization, mixed integer optimization is used. Using this procedure, a three-dimensional point set sampling a surface is mapped by a global chart onto a flat, two-dimensional domain. A similar approach, avoiding the costly computation of a reliable cut-graph on a point set is presented in [Jak+15].

This approach is now to a certain extend the contrary to the MLS approach for chart construction. While the latter creates extremely localized charts, the approach of Li et al. constructs a global parametrization on the topological disc obtained from the point set. In a practical application, it is however desirable to control the level of detail of the processed geometry. Thus, the user should be able to prescribe how many charts are needed for the application at hand. This cannot be done by these two approaches.

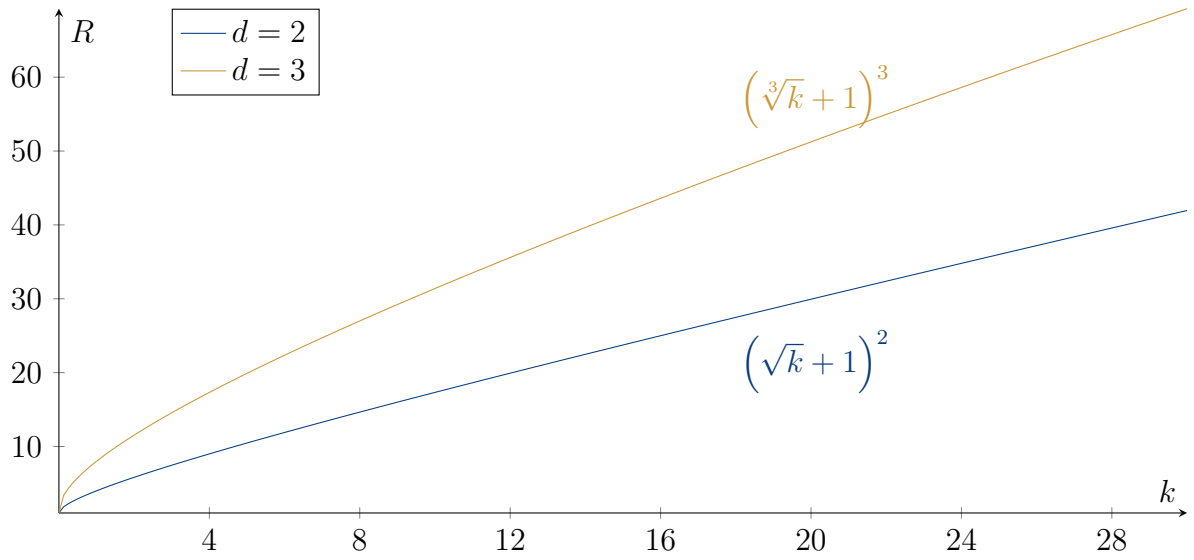


Figure 4.6: *Expected number of k-d tree cells R to be inspected when determining k neighbors of a point from a point set embedded in dimension d, cf. Equation 2.6.*

To make the local charts efficiently accessible, we furnish each of them with its own local data structure, e.g. a k-d tree (see Section 2) or a neighborhood grid (see Section 3). For the example of k-d trees, as proven in Section 2.2, the number of queries to be posed for neighborhood detection depends exponentially on the dimension of the space into which the point set is embedded. Given points distributed e.g. on a two-dimensional surface in $\mathbb{R}^3$, the height over the approximating hyperplane is negligible in contrast to the two-dimensional coordinates on the hyperplane. Therefore, in this case the queries can be sped up by only working on the chart domains, see Figure 4.6. To fully exploit this idea of locally flat structures, in the following, we aim at a technique that produces a variable number of locally flat coordinate charts in a point set.

5 Variational Shape Approximation

In the preceding section, we have reasoned that adaptive flat charts for point set surfaces are to be sought. Furthermore, in Section 1.3, we have presented a method to grow neighborhoods from the information given by the normal field. In order to construct flat regions in a point set, we turn to a similar idea. However, now, the growing process is not started from every point, but from a number of prescribed center points. Due to the variational problem involved, the approach was coined *Variational Shape Approximation* (VSA). The main contributions of the section are:

- Translation of the VSA algorithm to the setting of point sets.
- Enrichment of the algorithm with split and merge procedures to become independent of the number of prescribed centers.
- Explanation of an example showing the differences between the k means algorithm and the VSA procedure both on points and meshes.
- Description of a procedure to obtain a provable manifold structure from the set of grown flat regions for star-convex input.

5.1 Related Work

Our proposed method incorporates point set segmentation as well as a combination of surface reconstruction and simplification. Both will be addressed subsequently.

5.1.1 Segmentation

As we propose an algorithm working solely with point sets (based on the work of [CAD04] dealing with meshes and the work of [LB16] for point sets) we focus on segmentation algorithms in the same setting. An overview on mesh segmentation is presented by [Att+06]. The survey of [NL13] covers relevant developments for point set segmentation until 2013. In terms of their categorization, our method belongs to the area of seeded region-based methods, where we start the segmentation process by selecting seed points and let the regions grow by adding neighbors if they satisfy certain conditions like normal similarity. One early work in the same field is presented in [BJ88]. They start with a coarse segmentation to identify the seed points with the help of curvature labeling of points and then refine their regions while letting them grow iteratively according to attributes like proximity and planarity. The survey of [NL13] observes that the procedure of [BJ88] has difficulties with noise—as this heavily influences curvature—and is time consuming. The authors of [KS00] proposed a statistical approach with a focus on the merge of regions. They rely on the *mutual inlier ratio* (MIR) of adjacent regions and compute these using regression techniques and methods to estimate the robust scale of the Gaussian distribution. Their merging applies to an irregular graph pyramid storing relative information between the regions, which is utilized for comparison and merging. The authors of [TP05] use filtering to segment data obtained by laser scanning devices, to distinguish between terrain and object points. Here, they combine the strengths of two approaches. One which applies directly on the raw scanned data with the help of geometric attributes. The other first segments the data and classifies it according to the deduced segments. The region growing process is performed with respect to normal similarity and distance to a growing plane. In the article of [Sor+18], the authors also segment a 3D object into different charts. However, the main focus is on topological quantities. The work of [WI18] also presents a shape segmentation framework for point sets. It consists of a three-step procedure to (1) find a local shape description, (2) compute shape similarity measures, and (3) segment the shape from these. In particular the first step makes heavy

use of the discretized Laplace operator. The obtained result is however a semantic segmentation of the point set, as a human observer would segment it, see [CGF09]. Thus it differs from our approach as we aim for syntactically segmented point sets with as-similar-as-possible normals in each segment.

The works of [SWK07] and [AP10] both aim at fitting a set of primitives to the point set: plane, sphere, cylinder, cone, and torus. When compared to the VSA approach, these papers give similar results when reducing the primitive set to planes. Even then, the energy functionals fitting the planes to the point set are different. The authors of [SWK07] consider three points $p_1, p_2, p_3 \in P$ of their input point set P and compute a normal of the plane spanned by these points. This normal is then compared to the normals of p_1, p_2, and p_3 and the plane is considered to fit if all three normal variations stay below a user-given angle. In contrast, the authors of [AP10] place the plane at the weighted barycenter b of the segment $\{p_i\} \subset P$ to be considered. A weighted covariance matrix is used to determine the normal n and the fitting error is computed as

$$\sum_i w_i \left|\langle n, p_i - b\rangle_2\right|^2,$$

where w_i are the weight terms, see [AP10] for details. This neglects the normal information at the points p_i. Both works segment the point set semantically without regard for the local curvature information. Neither elaborates the surface reconstruction aspect.

In [LB16], the authors aim for feature extraction. Their reconstructed feature curves approximate the regions lying along them on both sides. As they set their focus on feature detection, they introduce a k-means algorithm similar to [CAD04] and the algorithm of this section, working on the input point set. In contrast to the presented work, they use a fixed number of proxies, instead of a fixed error threshold per proxy, which leads to the requirement of a good initial seeding. With our approach, we overcome this drawback of optimal seed positions in seeded-region based methods. The authors of [LB16] also give results of simplifications rendered as meshes without any analysis as to what requirements are needed in order to perform the surface reconstruction. To this end, we aim at a concise procedure to guarantee a closed manifold under consideration of combinatorial conditions in terms of the proxy neighborhoods. Furthermore, we give an explicit algorithm as well as its evaluation for the simplified surface reconstruction.

5.1.2 Surface Reconstruction & Simplification

A thorough introduction to the topic of surface reconstruction from point sets is given in the survey of [Ber+17]. The setting of point set representation is considered in [PKG06]. The latter present a multi-scale surface representation based on point samples. Given an unstructured point set as input, their method first computes a series of point-based surface approximations at successively higher levels of smoothness using geometric low-pass filtering. These point sets are then encoded relative to each other by expressing each level as a scalar displacement of its predecessor. Low-pass filtering and encoding are combined in an efficient multilevel projection operator using local weighted least squares fitting similar to [SL16].

Being precise, we neither reconstruct the surface in a general sense, as we only try to gain a coarse approximation of the surface based on the proxies instead of keeping all the details the point set might encode. Nor do we simplify the original point set, as we do not reduce the number of points. Simplification here can be understood as a way of coarsening a mesh—obtained from the given point set—to one with less elements while keeping the proxy information.

A brief introduction and (error) analysis of different point set simplification algorithms is given by [PGK02]. The authors summarize the types of clustering-methods and iterative methods as

well as particle simulation. With the help of an error measurement, they explain the advantages and disadvantages for all of the methods taken into consideration. The extreme simplification described by [RT09] concentrates on real-time rendering using points as primitives. These are realized utilizing oriented disks. Furthermore, they introduce a more complex primitive called *splat*, which approximates larger and more complex surface areas compared to the disks. Here, they also use a variant of segmentation, as they decompose the surface into quasi-flat regions by utilizing an efficient algebraic multi-grid algorithm. The authors of [XM09] work with simplification of dense point sets while keeping shape information, like features. The reduction process eliminates one point at a time by utilizing a k-d tree structure and local sampling density evaluation. Important attributes in real-world applications are the performance and quality of the rendering process. These are addressed in [SFC10]. They make use of an automatic adaption of splat sizes with respect to sampling density. The point reduction then utilizes a bilateral filtering algorithm, to simplify the point set while retaining features. A classical simplification algorithm for meshes is presented in [GH97], where the authors utilize quadric error metrics. This approach is combined with vertex clustering in the work of [HHL15]. We will use the procedure of [GH97] for comparison with our results.

5.2 The VSA Procedure

5.2.1 The VSA Procedure for Meshes

The VSA procedure was introduced by [CAD04]. It acts on a surface $S \subseteq \mathbb{R}^3$. The goal is to partition the surface into m disjoint regions $R_i \subseteq S, \dot{\bigcup} R_i = S$, where each region is associated a linear proxy $P_i = (C_i, N_i)$ with center $C_i \in \mathbb{R}^3$ and unit-length normal $N_i \in \mathbb{R}^3, i \in \{1, \dots, m\}$. The authors propose two different metrics to find the optimal shape proxies, with the first metric based on $\mathcal{L}^2$

$$\mathcal{L}^2(R_i, P_i) = \int_{x \in R_i} \|x - \pi_i(x)\|^2 \, dx, \tag{5.1}$$

where $\pi_i(.)$ denotes the orthogonal projection of the argument on the plane with normal N_i centered at C_i. Thus, the integral measures the squared error between points in the region R_i and its linear approximation given by P_i.

A second metric, denoted by $\mathcal{L}^{2,1}$ is based on the $\mathcal{L}^2$ measure on the normal field. It is given by

$$\mathcal{L}^{2,1}(R_i, P_i) = \int_{x \in R_i} \|n(x) - N_i\|^2 \, dx,$$

where $n(x)$ denotes the normal of the surface at point $x \in S$.

In the discrete setting, the surface S is given by a set of (triangular) elements t_j, $j \in [T]$ and the centers C_i are initially found by randomly choosing a triangle t_j as center C_i. Therefore, the second formulation can be simplified to

$$\mathcal{L}^{2,1}(R_i, P_i) = \sum_{t_j \in R_i} \|n(t_j) - N_i\|_2^2 \cdot |t_j|, \tag{5.2}$$

with $n(t_j)$ the normal and $|t_j|$ the area of the element t_j. As the authors conclude that the $\mathcal{L}^{2,1}$ metric is more effective, we will reduce the following discussion to this formulation.

The actual minimization of $\mathcal{L}^{2,1}$ with respect to the segmentation of S into the regions R_i and with respect to the proxies P_i is then performed iteratively. For this, a variation of Lloyd's fixed

point iteration [Llo82] is used. The first step is to pick a user-given number m of center elements $C_1, \dots, C_m$ randomly from the set of triangles $\{t_j \mid j \in [T]\}$. The normals N_i are set to the normals of corresponding center triangles C_i. The neighbors of the center triangles are collected in a priority queue $\mathcal{Q}$ sorted increasingly with growing $\mathcal{L}^{2,1}$ distance between neighboring triangle t_j and center triangle C_i resulting in $\|n(t_j) - N_i\|^2$. Then, the following three steps are performed iteratively until convergence[3]:

1. *Flood*: As long as the queue $\mathcal{Q}$ is not empty, pop the first element t_j from $\mathcal{Q}$. Ignore it, if it has already been assigned to a proxy. If it is not assigned yet, assign it to the proxy P_i that pushed it into the list and push all neighboring elements t_ℓ of t_j into $\mathcal{Q}$, noting that they have been pushed by P_i. Without loss of generality, we assume S to be connected. If that is not the case, the algorithm can simply be run on each connected component of S. For a connected surface S, after the queue $\mathcal{Q}$ has been emptied, all elements $\{t_j \mid j \in [T]\}$ have been assigned to some proxy P_i.

2. *Proxy Update*: The proxy normals N_i are updated according to
$$N_i = \frac{\sum_{t_j \in R_i} |t_j| n(t_j)}{\left\| \sum_{t_j \in R_i} |t_j| n(t_j) \right\|},$$
where $|t_j|$ denotes the area of the triangle t_j.

3. *Seed*: For all proxies P_i, find some $t' \in R_i$ such that $\|n(t') - N_i\|^2 \leq \|n(t_j) - N_i\|^2$ for all $t_j \in R_i$ and set $C_i = t'$. This ensures that the flooding in the next iteration is started from regions that best reflect the current proxy normals[4].

Finally, from the converged proxies, a simplified mesh is constructed.

5.2.2 The VSA Procedure on Point Sets

We will now proceed to translate the VSA procedure to the setting of point sets. Compared to the VSA on meshes, several details have to be adjusted for the method to work on point sets. At first, consider the partition problem as stated in Section 5.2.1. In the context of point sets, not elements, but the points themselves have to be assigned to the proxies. That is, the given point set $P = \{p_1, \dots, p_n\}$ will be partitioned into disjoint sets $\dot{\bigcup}_{i=1}^{m} R_i = P$. Therefore, in the following expressions, the centers C_i denote points from the point set, while the normals N_i at the respective center point are those obtained from the point set. The normals of the points p_j will be denoted by n_j respectively. As before, by P_i we denote the tuple $P_i = (C_i, N_i)$ and the N_i are assumed to have unit length.

Consider the energy as defined in Equation (5.2). For proxies defined from point sets, the area term $|t_j|$ cannot be used. Thus, we replace it by a weighting term which is to approximate the area represented by the point. We obtain the following energy of a single proxy and the energy formulation on the set of all proxies

$$\mathcal{L}^{2,1}(R_i, P_i) = \sum_{p_j \in R_i} \omega_j \|n_j - N_i\|^2, \tag{5.3}$$

$$E(\{(R_i, P_i) \mid i = 1, \dots, m\}) = \sum_{i=1}^{m} \mathcal{L}^{2,1}(R_i, P_i). \tag{5.4}$$

[3] See Section 5.2.3 for some remarks on convergence properties.
[4] This choice of seeds can however be problematic. See Section 5.2.3 for an example.

For our experiments, we simply use uniform weights $\omega_j = 1$ as these already provide good results. An approximation of the area term could be computed by $\omega_j = \|p_i - p_j\|_2^2$. Furthermore, bilateral or even binary weights can be employed.

The initial seeding as outlined above can still be done, but instead of triangles, now points $p_j \in P$ are chosen for the initial position of the center points C_i. Also, those points from P are initially pushed to the priority queue $\mathcal{Q}$ that are neighbors, but not identical, to the chosen center points C_i. For this neighborhood relation, any neighborhood concept such as k nearest (1.1) or Euclidean neighborhoods (1.4) can be used. Again, the points p_j in $\mathcal{Q}$ are sorted increasingly with $\mathcal{L}^{2,1}$ distance between their own normal n_j and the normal N_i of the proxy P_i that pushed them onto the queue: $\|n_j - N_i\|^2$. The following three steps remain almost unchanged:

1. *Flood*: As long as the queue $\mathcal{Q}$ is not empty, pop the first element p. Ignore it, if it has already been assigned to a proxy. If it is not assigned yet, assign it to the proxy P_i that pushed it into the list and push all neighboring elements p_j of p into $\mathcal{Q}$, noting that they have been pushed by P_i. As we assume S to be connected under the chosen neighborhood relation, after the queue $\mathcal{Q}$ has been emptied, all elements of P have been assigned to some proxy P_i.

2. *Proxy Update*: The proxy normals N_i are updated according to

$$N_i = \frac{\sum_{p_j \in R_i} \omega_j n_j}{\sum_{p_j \in R_i} \omega_j}. \tag{5.5}$$

3. *Seed*: For all proxies P_i, find some $p_\ell \in R_i$, $\ell \in [n]$, such that $\|n_\ell - N_i\|^2 \leq \|n_j - N_i\|^2$ for all $p_j \in R_i$.

5.2.3 Convergence of the VSA Procedure

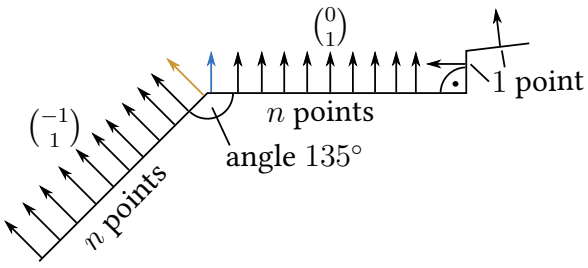
Concerning the convergence of their algorithm, the authors of [CAD04] state:

> *(...) Lloyd's algorithm always converges in a finite number of steps, since each step reduces the energy E: the partitioning stage minimizes E for a fixed set of centers c_i, while the fitting stage minimizes E for a fixed partition.*

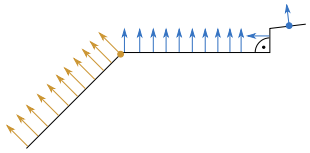
While this statement holds for the original algorithm of Lloyd as presented in [Llo82], it does not hold for neither the VSA procedure on meshes nor on point sets as the following example shows. Consider the two-dimensional setup shown in Figure 5.1a. It is given by n points connected on a line with normal $\binom{-1}{1}$ next to a line of n points with normal $\binom{0}{1}$. At the right end of the second line, there is a single point with normal $\binom{-1}{0}$ and another single point with normal N given by the equation

$$N = \frac{1}{n+2}\left(n \cdot \begin{pmatrix}0\\1\end{pmatrix} + \begin{pmatrix}-1\\0\end{pmatrix} + N\right).$$

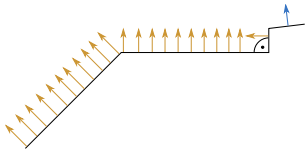
Now, two proxies will act on this example, with their initial seeds shown in yellow and blue in Figure 5.1a. They each start on one of the two lines of n points respectively. The result after a flood is shown in Figure 5.1b, where each line is completely covered by the proxy starting on it and the two single points are associated to the proxy with normal $\binom{0}{1}$. After updating the proxy normals, the yellow proxy has normal $\binom{-1}{1}$ while the blue proxy has normal N given by the



(a) *Setup for growing error functional.*



(b) *Segmentation after first flood.*



(c) *Segmentation after second flood.*

equation above. Thus, the yellow proxy starts from an arbitrary point on its line while the blue proxy starts from the rightmost point. The error after this first flood and proxy update is given by

$$E_1 = n \cdot \left\| \begin{pmatrix} 0 \\ 1 \end{pmatrix} - N \right\|^2 + \left\| \begin{pmatrix} -1 \\ 0 \end{pmatrix} - N \right\|^2 .$$

Starting from the new seed points, a second flood results in the situation shown in Figure 5.1c. Here, almost all points except for the rightmost one are associated to the yellow proxy with normal $\binom{-1}{1}$. Its new normal after a proxy update is

$$N' = \frac{1}{2n+1} \left(n \cdot \begin{pmatrix} -1 \\ 1 \end{pmatrix} + n \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} -1 \\ 0 \end{pmatrix} \right),$$

which amounts to an error after the second flood and proxy update given by

$$E_2 = n \cdot \left\| \begin{pmatrix} -1 \\ 1 \end{pmatrix} - N' \right\| + n \cdot \left\| \begin{pmatrix} 0 \\ 1 \end{pmatrix} - N' \right\| + \left\| \begin{pmatrix} -1 \\ 0 \end{pmatrix} - N' \right\|.$$

Choosing $n = 100$, we obtain $E_1 = 1.9802$, but $E_2 = 39.395$. Furthermore, the corresponding error value after the flood is also growing. Thus, the statement of on always shrinking error after each combined flood and proxy update procedure is incorrect.

However, we can prove the following statement.

Theorem 14 (Error reduction by proxy update, M. S. and E. Zimmermann)**.** *Given a point set $P = \{p_1, \ldots, p_n\}$ with a neighborhood structure, such that the neighborhood graph on P is connected and normals $n_1, \ldots, n_n$ on P, then each proxy update step as defined above leads to proxies (R_i, P_i) with a smaller error measure in Equation (5.4).*

Proof. Consider

$$\nabla E(\{R_i, P_i\}) = \nabla \left(\sum_{i=1}^{m} \mathcal{L}^{2,1}(R_i, P_i) \right) = \sum_{i=1}^{m} \sum_{p_j \in R_i} \nabla \omega_j \left\| n_j - N_i \right\|^2 = \sum_{i=1}^{m} \sum_{p_j \in R_i} 2\omega_j (n_j - N_i).$$

Setting $N_i = \frac{\sum_{p_\ell \in R_i} \omega_\ell n_\ell}{\sum_{p_\ell \in R_i} \omega_\ell}$ as in Equation (5.5), we obtain

$$\begin{aligned} \sum_{p_j \in R_i} 2\omega_j (n_j - N_i) &= \sum_{p_j \in R_i} 2\omega_j n_j - \sum_{p_j \in R_i} 2\omega_j \left(\frac{\sum_{p_\ell \in R_i} \omega_\ell n_\ell}{\sum_{p_\ell \in R_i} \omega_\ell} \right) \\ &= \sum_{p_j \in R_i} 2\omega_j n_j - \left(\frac{\sum_{p_\ell \in R_i} 2\omega_\ell n_\ell}{\sum_{p_\ell \in R_i} \omega_\ell} \right) \cdot \sum_{p_j \in R_i} \omega_j \\ &= \sum_{p_j \in R_i} 2\omega_j n_j - \sum_{p_\ell \in R_i} 2\omega_\ell n_\ell = 0. \end{aligned}$$

Thus, at the chosen updated proxy normal, the energy reaches a (local) minimum. As the energy is convex as sum of norms, which are convex, the found minimum is indeed its global minimum for the current choice of segmentation. □

It remains to be seen in further research whether the convergence of the algorithm can be shown theoretically or whether the algorithm can be altered in order to achieve provable convergence.

5.2.4 User controlled Level of Detail

With the algorithm, we aim at adaptability of the constructed flat pieces towards user input. That is, the user should be able to control the level of detail obtained from the flat regions. In addition to the three-step procedure given in the preceding section, we will include two more steps which work towards the goal of adaptability. For this, we introduce a user-given parameter $\kappa \in \mathbb{R}_{\geq 0}$ which controls the maximum deviation from its flat approximation within a proxy region R_i. This parameter is used in the following two steps:

(a) *Split*: Given a proxy P_i with its region R_i such that $\mathcal{L}^{2,1}(R_i, P_i) > \kappa$. We use weighted principal component analysis by [HBC11] to compute the most spread direction of R_i. The set R_i is then split at the center of this direction into two new regions $R_i = R_i^1 \dot{\cup} R_i^2$. The new normals are chosen as $N_i^1 = \sum_{p_j \in R_i^1} \frac{\omega_j n_j}{\sum_{p_j \in R_i^1} \omega_j}$ and as a corresponding N_i^2 respectively. The new centers C_i^1 and C_i^2 are then placed at those points of R_i^1, R_i^2 that have least varying normals from N_i^1 and N_i^2 respectively.
Note that performing a split operation reduces Energy (5.3) and thus speeds up the convergence of the algorithm.

(b) *Merge*: Consider a pair P_i, P_j of neighboring (see definition above) proxies with their respective normals N_i, N_j. If the region $R' = R_i \cup R_j$ with normal $N' = \frac{N_i + N_j}{2}$ achieves an Energy (5.3) strictly less than κ, the two regions are replaced by their union R', with normal N' and a chosen center $C' \in R'$ with its normal least deviating from N'.
Note that we could allow only those pairs of neighboring regions to merge such that

$$\mathcal{L}^{2,1}(R_i, P_i) + \mathcal{L}^{2,1}(R_j, P_j) \leq \mathcal{L}^{2,1}(R', P').$$

Then, the Energy (5.4) would not increase and termination of the algorithm would be guaranteed. However, this would result in neighboring regions not observing the user-given κ threshold. Therefore, we accept an increase of the global energy in favor of a better region layout. In all experiments, the algorithm still converged.

Both operations alter the number m of proxies. Thereby, a significant disadvantage of the algorithm of [Llo82] is eliminated as the user does not have to choose m a priori. It is replaced by the user's choice of κ, providing a semantic guarantee on the regions being built by the algorithm. The user can prescribe a value of κ computed from the desired curvature within a proxy.

The possible presence of noise in the point set P gives yet another reason to refute Energy (5.1). For points distributed around the xy-plane, with normals $(0, 0, 1)^T$ and just slight deviation from the plane, this energy would create larger values for a growing number of points, while the Energy (5.4) does not suffer from this. Hence, with the chosen energy, noise on the point positions is handled more robustly.

In the merge process outlined above, we asked for two neighboring regions. However, we have not defined any relation on the regions yet. In the meshed case discussed in Section 5.2.1, two

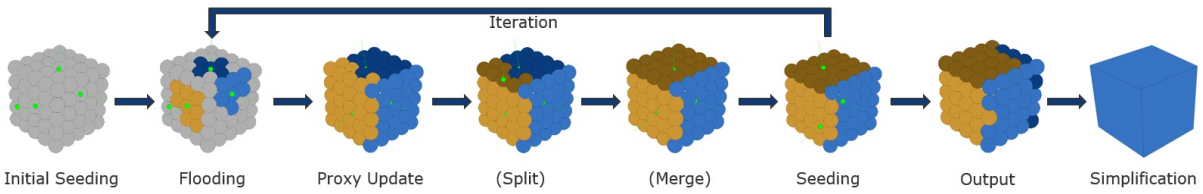


Figure 5.2: *The whole pipeline contains an initial random seed selection, an iteration over flooding, proxy updates, possible splits and merges of regions, and a search for new seeds. The output is reached after convergence or after a fixed number of iterations. Afterwards, we deduce a simplified model according to the proxies, which can also be considered as a surface reconstruction from the initial point set.*

regions are neighbors if and only if they share an edge in the mesh. In the context of point sets, we do not have any information on connectivity. Thus, we propose the following definition. During the flooding step described above, an element p is popped from the priority queue $\mathcal{Q}$ together with a possible assignment to a region R_i. However, it is ignored if p has already been assigned to a region R_j. In this case, we denote R_i and R_j to be neighbors, if $i \neq j$.

This finishes the whole VSA pipeline for point sets, including the additional two steps *merge* and *split*. See Figure 5.2 for an illustration of the complete pipeline.

5.3 Simplified Shape Reconstruction

In the preceding Sections 5.2.2 and 5.2.4 we have described how the input point set P can be segmented into flat proxy regions R_i. Starting from a corresponding structure on meshes, the procedure in [CAD04] simplified the underlying mesh until only one element for each proxy is left. In this Section, we aim at a similar result. This is more involved however, as the input point set in our case does not provide the connectivity given by meshes. Furthermore, neither [CAD04] nor [LB16] provide detailed descriptions on how to perform the shape simplification or reconstruction from the identified proxies. With this section, we close this gap.

While the above sections can be applied to any point set, in the following, we will assume that the input point set P is a sample of a 2-manifold $\mathcal{M}$. For the sake of simplicity, we assume the manifold $\mathcal{M}$ to be closed and embedded in $\mathbb{R}^3$, which is not necessary for the following to work. Recall that a triangulation of the topological space $\mathcal{M}$ is a simplicial complex K, homeomorphic to $\mathcal{M}$, together with a homeomorphism $h : K \to \mathcal{M}$. In this case, the triangulation is sometimes referred to as simplicial manifold. An immediate consequence to this definition is that the star of each vertex of K is homeomorphic to a 2-dimensional disk. This, in turn is equivalent to the following two conditions:

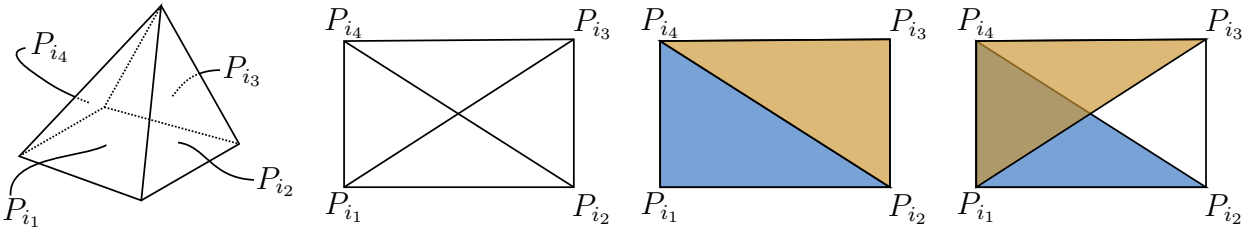
1. Every edge of the complex K is incident to exactly two 2-simplices (as we only consider closed manifolds).

2. The vertex star of each vertex is unique and connected.

On these notions, we will build a shape reconstruction algorithm.

5.3.1 Implementation

From Sections 5.2.2 and 5.2.4 we have a set of proxies $P_i = (C_i, N_i)$ with centers C_i and unit-length normals N_i, $i \in [m]$. Intuitively, it is necessary to introduce a line where two proxies P_i and



(a) *Four Proxies identified as the sides of a pyramid.* **(b)** *Neighborhood graph of the four proxies.* **(c)** *One possibility to split into two triangles.* **(d)** *Illegal break into two triangles.*

Figure 5.3: *(a) Around the tip of a pyramid, four proxies $P_{i_1}, P_{i_2}, P_{i_3}, P_{i_4}$ are identified. (b) The neighborhood graph on these four proxies has the connectivity of a tetrahedron. In order to project it to a two-dimensional simplicial manifold, two triangles have to be deleted. (c) A covering with two triangles that make the simplicial complex a simplicial manifold. (d) When deleting the wrong triangles, the resulting complex K is not a closed simplicial manifold anymore.*

P_j meet and a vertex where three proxies meet. However, the neighborhood relation presented and used at the end of Section 5.2.4—which only focuses on two proxies P_i and P_j being neighbors to each others—will not lead to a satisfactory reconstruction of the shape. This can be seen by the following easy example. Consider a prism over a triangle, with three rectangular sides F_1, F_2, F_3. Then, a line will be introduced for each pair (F_1, F_2), (F_2, F_3), and (F_1, F_3). As the three sides are pairwise neighbors, a vertex is to be introduced at the intersection point, but this point lies at infinity.

Therefore, we work with the following alternative concept. First, we start with an empty simplicial complex $K = \emptyset$. Now, we iterate over all points $p \in P$ of the original input point set. For each point, note to which proxies P_i the points in its neighborhood are assigned. If the number of proxies witnessed in the neighborhood is less than three, nothing is done. Assume that there are points in the neighborhood of p assigned to proxies $P_{i_1}, \ldots, P_{i_\ell}$, then we add the set $\{i_1, \ldots, i_\ell\}$ to K. After iterating over all $p \in P$, we clean K by removing all sets $S \in K$, where $\exists S' \in K$ with $S' \neq S$ and $S \subseteq S'$. Thereby, we only keep the largest sets in K.

Finally, consider an element $S \in K$ with $|S| \geq 4$. Then, at some point $p \in P$, at least four proxies have been witnessed in the neighborhood. As long as any S with $|S| \geq 4$ exists in K, the simplicial complex is not a simplicial manifold as defined above. Thus, for each such S, we replace it by sets of triples. Consider for example $S = \{i_1, i_2, i_3, i_4\} \in K$. Then, we remove S from K, but replace it by $\{i_1, i_2, i_4\}$ and $\{i_2, i_3, i_4\}$ which are added to K. Here, we have to be very careful, however. The situation is depicted in Figure 5.3. Note that the wrong selection of triangles will lead to a configuration which does not provide a simplicial manifold, see Figure 5.3d.

In order to prevent illegal configurations of triangles, we perform the following. First, we build the average $\bar{N}$ over all normals $N_{i_1}, N_{i_2}, N_{i_3}, N_{i_4}$ of the proxies $P_{i_1}, P_{i_2}, P_{i_3}, P_{i_4}$ respectively to prevent loss of information during the following projection. We further consider the barycenter $\bar{C}$ of the center points $C_{i_1}, C_{i_2}, C_{i_3}, C_{i_4}$. Now, we project the centers $C_{i_1}, C_{i_2}, C_{i_3}, C_{i_4}$ onto the hyperplane spanned by $\bar{N}$ at $\bar{C}$. We sort the projections around p and thus ensure that for four proxies, always a situation as in Figure 5.3c—as opposed to the situation of Figure 5.3d—is chosen. Furthermore, we ensure that the line of intersection between two opposite proxy planes of P_{i_1} and P_{i_3} is closer to $\bar{C}$ than the intersection of P_{i_2} and P_{i_4}. Thereby, the reconstruction will not intersect itself. We proceed similarly with all sets of K that have higher cardinality than 3 and thereby obtain a simplicial complex that is solely comprised of triple sets.

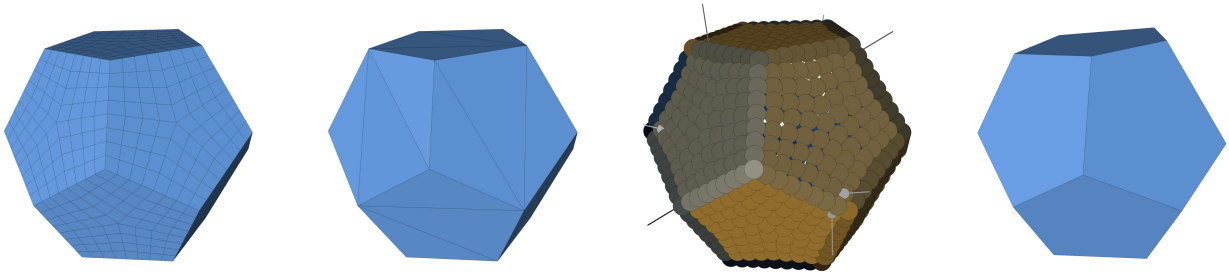
The final reconstruction is then done in two easy steps. First, we create a vertex for each triple

$(P_{i_1}, P_{i_2}, P_{i_3})$ which is placed at the intersection of the three proxy planes. Second, for each proxy P_i, we collect all newly created vertices from triples in which P_i has listed. These vertices are projected onto the plane spanned by N_i at C_i and sorted around the center C_i. Then, a new element is created from the new vertices with the obtained sorting. The whole procedure is given in Algorithm 4.

Algorithm 4 Simplified Surface Reconstruction from Proxies $\{P_i\}$

procedure RECONSTRUCTION(Proxies $\{P_i\}$)
  $K \leftarrow \emptyset$
  for all points $p \in P$ **do**
    if number of different proxies encountered in neighborhood of p is > 2 **then**
      $K \leftarrow K \cup \{$indices of proxies encountered$\}$
    end if
  end for
  for all $S \in K$ **do**
    if $\exists S' \in K : S \neq S', S \subseteq S'$ **then**
      $K \leftarrow K \backslash \{S\}$
    end if
  end for
  for all $S \in K$ where $|S| > 3$ **do**
    Delete S from K but introduce a sufficient number of non-overlapping triple sets $\{i_1, i_2, i_3\}$ to cover S.
  end for
  for all $(i_1, i_2, i_3) \in K$ **do**
    Create a vertex v_j as intersection of $P_{i_1}, P_{i_2}, P_{i_3}$.
  end for
  for all proxies P_i **do**
    Collect all vertices v_j that come from a triple involving i
    Project these vertices to the hyperplane spanned by C_i and N_i
    Order the projected vertices around C_i and create an element from this order representing P_i.
  end for
end procedure

Note that the reconstruction algorithm in its current form is limited to an element for proxy P_i that is star-convex with respect to its center C_i, that is, that the line from C_i to any point in the element of P_i is fully included in the element of P_i. A more involved and thus capable algorithm could reconstruct the vertices as described, but then use those points of P associated to P_i and e.g. a boundary detection on point sets, see e.g. [MPS19]. Thereby, also non-star-convex regions can be identified and represented by a single element. A corresponding algorithm is clearly available to [LB16] as proxies in their applications obtain non-star-convex elements. However, in any case, the construction of vertices as described above assures that the overall reconstructed surface is a simplicial manifold. This also holds for alternative ways of connecting the vertices to the final elements.



(a) *Original mesh of the dodecahedron.*

(b) *Simplified mesh by [GH97].*

(c) *Segmentation by the proposed method.*

(d) *Output of the proposed method.*

Figure 5.4: *The dodecahedron model and its two simplifications.*

model	s	min	max	mean	RMS	Method
Dodecahedron	2886	0.000000	0.000000	0.000000	0.000000	GH
	2886	0.000000	0.077600	0.023339	0.029738	Ours
Octahedron	4640	0.000000	0.000000	0.000000	0.000000	GH
	4640	0.000001	0.034809	0.009724	0.011725	Ours
Bullet	2325	0.000000	0.205391	0.030992	0.047718	GH
	2325	0.000000	0.198509	0.036149	0.050088	Ours
Moai	30006	0.000001	0.759781	0.152552	0.193410	GH
	30006	0.000008	0.766045	0.158608	0.221292	Ours

Table 5.1: *Results of the comparisons between the algorithm of [GH97] (GH) and our proposed method. Performed with the* Metro *algorithm of [CRS98]*

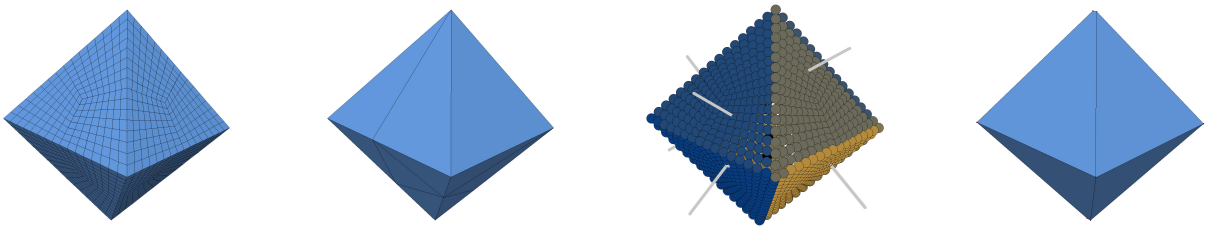
5.4 Experimental Results

In order to evaluate the output of our algorithm, we compare it to the mesh simplification by [GH97]. The input to all experiments is a mesh, see e.g. Figure 5.4a. For our algorithm, the elements and edges are ignored and the point set without any connectivity information is used. We obtain a reconstructed surface, see e.g. Figure 5.4d. For this surface, we count the number of elements and compute the number of triangles necessary (as our simplification also contains n-gons with $n > 3$). Finally, we apply the algorithm of [GH97] and prescribe the obtained number of triangles as target, see e.g. Figure 5.4b.

To compare the two simplifications obtained by [GH97] and our proposed method, we turn to the *Metro* algorithm of [CRS98] which is available as part of [Cig+08]. The *Metro* algorithm is run from the respective original mesh of the model to the different simplifications. It computes the Hausdorff distance of the two meshes based on a sampling approach. In Table 5.1, we give the number of samples (s), the minimum distance ($\min$), the maximum distance ($\max$), the mean distance (mean), and the root mean square distance (RMS). The used models are the Dodecahedron, Figure 5.4, the Octahedron, Figure 5.5, the Bullet model, Figure 5.6, and the Moai model, Figure 5.7.

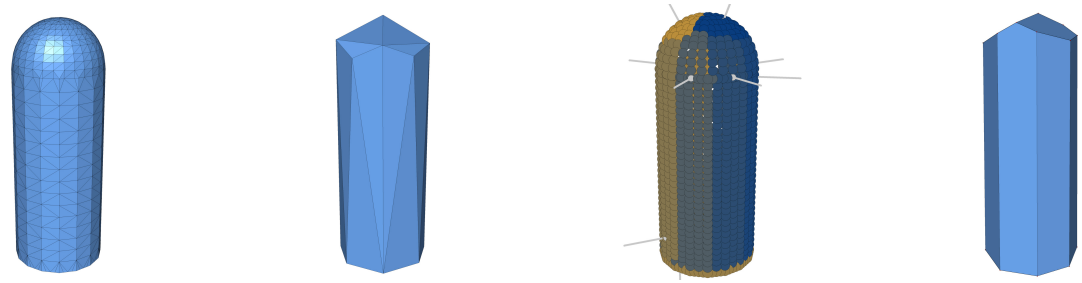
From the results presented in Table 5.1, it becomes clear that the proposed method performs slightly worse than the simplification of [GH97]. However, it does not have explicit connectivity information at hand and the differences are comparably small. Thus, the algorithm provides reliable results for simplification of point sets.

We proceed by investigating the effect of κ on the output of the algorithm. Different values for κ and the corresponding results on a single geometry are shown in Figure 5.8.



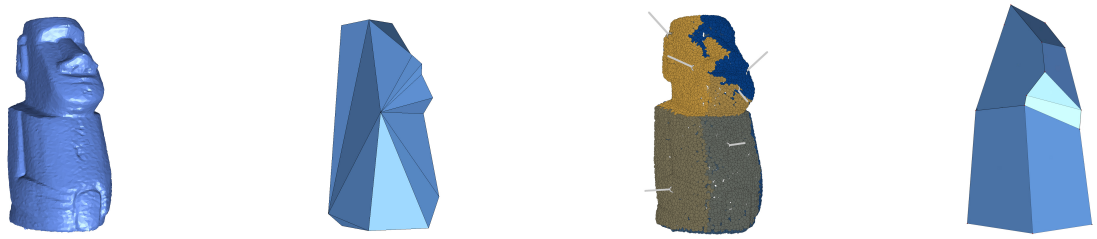
(a) *Original mesh of the octahedron model.* **(b)** *Simplified mesh by [GH97].* **(c)** *Segmentation by the proposed method.* **(d)** *Output of the proposed method.*

Figure 5.5: *The bullet model and its two simplifications.*



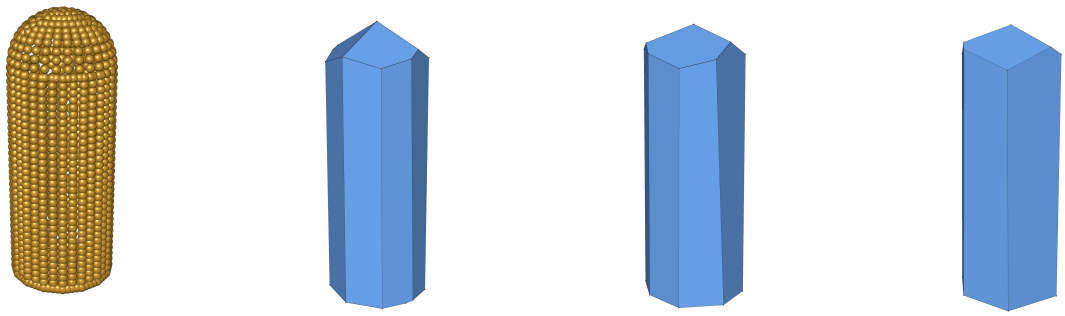
(a) *Original mesh of the bullet model.* **(b)** *Simplified mesh by [GH97].* **(c)** *Segmentation by the proposed method.* **(d)** *Output of the proposed method.*

Figure 5.6: *The bullet model and its two simplifications.*



(a) *Original mesh of the Moai model.* **(b)** *Simplified mesh by [GH97].* **(c)** *Output of the proposed method.* **(d)** *Output of the proposed method.*

Figure 5.7: *The Moai model and its two simplifications.*



(a) *Original point set of the bullet model.* **(b)** $\kappa = 10$, 17 *elements* **(c)** $\kappa = 20$, 13 *elements* **(d)** $\kappa = 30$, 11 *elements*

Figure 5.8: *The effect of different values for* κ *on the bullet model. Note that all results were computed completely automatically without prescribed number of proxies or manual selected seeds.*

All results presented in this section are produced from an initial random seeding, while the results from [CAD04] and [LB16] are in part dependent on a well-chosen manual seeding. Yet, our results are visually comparable to those presented in [CAD04] and [LB16]. Therefore, it can be seen that our method is independent of manual initial seed placing and can provide similar results automatically.

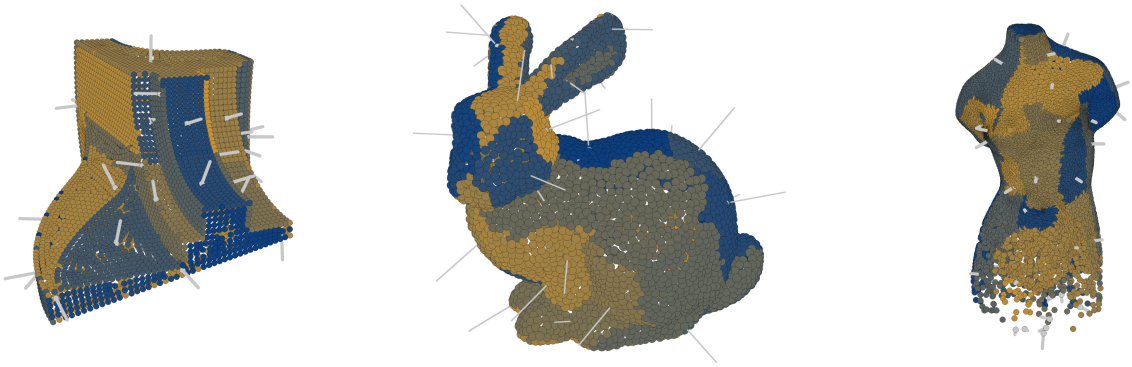
5.5 Conclusion

In this section, we have presented an adaption of the algorithm of [CAD04] to the setting of point sets. In addition to the algorithm of [LB16], we have added two steps to the pipeline. Namely, a *merge* and a *split* step. By these additional steps, our procedure is independent of an a priori given number of proxy regions. Furthermore, it is independent of manually selected seeds, as we have shown in our experiments. Additionally, we provide an example to show that the assumption made in previous publications—the decrease of Energy (5.4) for each flood and update—is incorrect. This poses the question of a full theoretical convergence and termination analysis of the algorithm which is left as future work.

Concerning surface reconstruction and simplification, we provide a thorough theoretical understanding of the reconstruction of a simplified mesh from the proxy set. Our theory is rooted in the context of simplicial complexes and guarantees the reconstruction to be a closed combinatorial manifold surface. A quantitative analysis on the results of the proposed algorithm in comparison with the simplification of [GH97] shows that our algorithm performs only slightly worse despite the lack of any connectivity information or area weighting.

However, our procedure is currently restricted to star-convex input. Consider Figure 5.9 for an illustration of problematic cases. Namely, in the fandisk model, Figure 5.9a, two proxies are created at the front of the model. These proxies are almost co-planar and have an intersection far our of the model. Thus, our surface reconstruction algorithm introduces several points not in the vicinity of the original input. In the ear of the bunny model, Figure 5.9b, a proxy is completely surrounded by another proxy. As we ask for at least three proxies to come together to form a vertex, this proxy does not have enough neighboring proxies to be considered in the surface reconstruction process. Thus, its points will be neglected. Finally, the Venus model, Figure 5.9c, showcase problems with varying density. In the lower parts of the model, where sampling density is very low, several proxies are introduced on few points which does not follow the idea of collecting large, piecewise flat regions.

Aside from these problems, three questions remain open and are left for further research. First,



(a) *Segmentation of the fandisk model obtained by the proposed method. Note the two almost co-planar proxies at the front meeting far out the model (not shown).*

(b) *Segmentation of the bunny model obtained by the proposed method. Note the proxy region in the ear of the bunny which is completely surrounded by another proxy and thus disregarded in our surface reconstruction as we ask for three proxies to come together and form a vertex.*

(c) *Segmentation of the Venus model obtained by the proposed method. Note how the low sampling density in the lower parts of the model create several proxies only consisting of very few points.*

Figure 5.9: *Three models segmented with the proposed method that yield problematic areas.*

it is unclear how the algorithm behaves on a noisy point sample. As this is also unclear for the algorithms of [CAD04] and [LB16], the effect of noise calls for a separate investigation. Second, our example showed that Energy (5.4) is not always shrinking with the proposed procedure. Therefore, convergence and termination of the algorithm is not given and has to be established in future work. A third and final questions concerns the weights ω_j used in Equation (5.3). Currently, as results are already good, we used equal weight $\omega_j = 1$. How exactly the choice of different weights effects the output of the algorithm is left for further research.

III Robust and Efficient Processing of Point Sets

6 Directional Density Measure to Intrinsically Estimate and Counteract Non-uniformity in Point Sets

When considering point sets sampling surfaces embedded in $\mathbb{R}^3$, many algorithms make additional assumptions on the sampling quality. For example, Amenta et al. assume in their power crust algorithm, see [ACK01], that the distance of the sample to the sampled object is bounded by the local feature size. Results that are independent of the sampling—like the k-d tree data structure of Friedmann et al., see [FBF77], discussed in Section 2—are somewhat rare. In particular, when discretizing differential geometry operators like the gradient, the Laplace operator, or the shape operator [BSW09; LP05; Tau95], implicit or explicit assumptions are made on the uniformity of the point set.

However, many point sets that arise in applications do not satisfy the assumptions made in theory. One can think of the manual scanning of some object by a hand-guided laser scanner. The operator of the scanner will recognize problematic areas—for example pits, areas of high curvature, or other features—and scan these multiple times in order to capture them accurately and without occlusion. Other less diverse areas are scanned fewer times. In the border regions between these two area types, regular euclidean neighborhoods will favor those areas that have been scanned multiple times (see Section 1 for a discussion of neighborhood concepts). That is because they simply have more points in the sample. Standard algorithms fail to process these regions correctly.

In this section, we present an approach to handle non-uniform densities in point sets. Our method works intrinsically without using additional information beside the raw point set. We define a local density measure from a point of the point set into any tangential direction. The measure is transformed to give local weights on the neighborhood of each point. These are then used e.g. in the discretizations of discrete differential operators as mentioned above in order to make them robust even on non-uniform samples. Computational experiments show the effectiveness of our approach on both synthetic and real world data.
The main contributions of this section are:

- Definition of a discrete directional density measure for point sets.
- Intrinsic computation of local density without additional information.
- Definition of weights for discrete operators to overcome non-uniform density problems.

The results of this section have been presented at the GMP'18 conference and published in the corresponding proceedings, see list of publications prior to the thesis, page 5. Additional to the results published, this section includes an evaluation of the presented weights in the context of a blue noise sampling.

6.1 Related Work

Several methods have been devised to handle non-uniformly sampled point sets within different application scenarios. We will give a brief overview here.

In the setting of surface reconstruction, the authors of [HK06] propose an unsigned distance function which enables them to process input data consisting solely of 3D sample positions without any normal information. Since the surface extraction does not depend on a change of sign of the implicit representation, the method is immune to noisy and non-uniformly distributed samples. A different approach is taken in [LPZ13]. Here, the authors add volumetric or prior information to global implicit surface reconstruction to eliminate the ill-posedness of non-uniformly sampled point sets. Yet another proposal is made in [Yan+17], where a local hierarchical clustering method is used to improve the consistency of the point set distribution. Within a two-step process, the computational complexity of the point set is reduced and the remaining points are transformed into a uniform sampling. Similarly, the authors of [Mos+17] use a combination of octree data partitioning, local Delaunay tetrahedralization, and graph cut optimization for their surface reconstruction approach. They achieve processing of point density variations of more than four orders of magnitude.

For registration of point sets with non-uniform density, the authors of [HB14] propose to extend registration algorithms by including topological information on the sampled surface. Thereby, they aim at compensating effects that the non-uniformity has on 3D neighborhood searches. One of the main applications of [Che+17] is also registration. The authors use non-uniform density in order to reduce cost in storing, processing, and visualizing a large-scale point set. They consider a randomized resampling strategy to select a representative subset of points while preserving features depending on the application.

Considering simplification methods for point-sampled surfaces, in [PGK02], it is described that non-uniform samplings call for more sophisticated approaches yielding higher computational overhead. Furthermore, the authors state that finding suitable global factors can be difficult for non-uniformly sampled point sets.

The general problem of non-uniformity in point sets is approached in [WW11]. Finding that analyzing methods for uniform samples cannot be easily extended to non-uniform settings, the authors present an extension of Fourier analysis to measure spectral and spatial properties of various non-uniform sample distribution. This includes in particular adaptive, anisotropic, and non-Euclidean domains. A different approach is taken in [ÖAG10]. The authors use spectral analysis of manifolds to derive optimal sampling conditions for a given surface representation. However, they also find that if their method is directly used on a point set with a non-uniform distribution, limitations on convergence and stability may arise.

A different approach is taken in [Cue14; Cue+15; CLT14]. The authors investigate estimators of normals and the Voronoi covariance measure, for which they derive several stability results. These are favorably compared to the state of the art as the estimators are robust to both noise and outliers. This approach seems to be somewhat orthogonal to the ideas presented here. Thus, a connection of both ideas could yield a highly stable estimator.

In conclusion, the state-of-the-art methods either propose solutions tailored to a special application, try to deal with non-uniformity by adding additional data, or do not work reliably on non-uniform point samples at all. In the following, we present an approach that is general, does not need further information beside the point set itself, and can handle non-uniform samples.

6.2 Three Approaches to Directional Density Measures

We will now introduce three different directional density measures on point sets. The first is a continuation of the work presented in [LP05] and builds on the covariance matrix. The second works with a unit circle on the tangent plane and the corresponding segments induced by projections of neighboring points. Finally, the third measure utilizes smooth basis functions.

We consider a point set $P = \{p_i \in \mathbb{R}^3 \mid i = 1, \dots, n\}$. Denote for each point $p_i \in P$ its neighborhood by $\mathcal{N}_i \subseteq P$. We will assume $p_i \notin \mathcal{N}_i$. Note that the following considerations are independent of the actual definition of neighborhood. Therefore, they can be applied for e.g. a combinatorial k nearest approach as well as for Euclidean neighborhoods (see Section 1 for a discussion of neighborhood concepts).

In general, it is desirable to have a neighborhood that is as symmetric as possible with regard to all directions. When using combinatorial neighborhoods with k nearest neighbors, points lying almost as close to the query point as others might be excluded simply because k neighbors have already been found. Furthermore, to prevent numerical errors, especially when dividing by the distance, we drop those points that lie at most ε_m away from the query point, where ε_m denotes the machine accuracy. We define the k nearest neighborhood of $p_i \in P$ with distance at least ε_m by

$$\tilde{\mathcal{N}}_i = \{p_j \in P \mid j = 1, \dots, k,\ p_j \text{ the } k\text{-nearest neighbors to } p_i \text{ with } \varepsilon_m < \|p_i - p_j\|\}.$$

This neighborhood is then relaxed to

$$\mathcal{N}_i = \tilde{\mathcal{N}}_i \cup \{p_j \in P \mid \operatorname{abs}(\|p_i - p_j\| - \max_{p_\ell \in \tilde{\mathcal{N}}_i} \|p_i - p_\ell\|) \leq \varepsilon_m\}, \tag{6.1}$$

where abs denotes the absolute value, see Equation (1.7). That is, we include all those neighbors, that did not fall into the k nearest neighborhood by a small account. This definition of neighborhood was proposed in [Jan17].

We denote by $C_i \in \mathbb{R}^{3\times 3}$ the covariance matrix of the set $\mathcal{N}_i$, given by

$$C_i = \sum_{p_j \in \mathcal{N}_i \cup \{p_i\}} (p_j - \bar{p}_i)(p_j - \bar{p}_i)^T, \qquad \bar{p}_i = \frac{1}{|\mathcal{N}_i| + 1} \sum_{p_j \in \mathcal{N}_i \cup \{p_i\}} p_j$$

with $\bar{p}_i$ the barycenter. Furthermore, we identify its eigenvalues by $\lambda_{i,1}, \lambda_{i,2}, \lambda_{i,3} \in \mathbb{R}$. Note that the covariance matrix is positive semidefinite, therefore all eigenvalues are nonnegative. We assume $\lambda_{i,1} \geq \lambda_{i,2} \geq \lambda_{i,3}$ and call the corresponding eigenvectors $v_{i,1}, v_{i,2}, v_{i,3} \in \mathbb{R}^3$, with $\|v_{i,1}\| = \|v_{i,2}\| = \|v_{i,3}\| = 1$.

The covariance matrix induces a tangent plane T_i at the neighborhood $\mathcal{N}_i$ with distance r to a center point b and normal vector n by minimizing the least squares energy

$$E(n, r) = \sum_{p_j \in \mathcal{N}_i \cup \{p_i\}} (\langle p_j - b, n\rangle - r)^2.$$

The minimum is obtained for: $b = \bar{p}_i$, n the eigenvector $v_{i,3}$ of C_i corresponding to the smallest eigenvalue $\lambda_{i,3}$, and $r = 0$, see [Skr14b, Section 2.1.2]. As $v_{i,1}$ and $v_{i,2}$ are of norm 1 and orthogonal, they form an orthonormal basis for T_i.

The final goal is to define weights on all neighboring samples $p_j \in \mathcal{N}_i$ of a point $p_i \in P$. These weights can then be used in any discretization scheme, e.g. for the gradient, the Laplace operator, or the shape operator [BSW09; LP05; Tau95]. A first expectation on the weights is non-negativity. Secondly, as a test case with known ground truth for our density measures, we consider the neighborhood $\mathcal{N}_i$ to be the vertices of a regular $|\mathcal{N}_i|$-gon, see Figure 6.1. In this regularly sampled case, each weight should be roughly 1. Therefore in the following, we normalize all presented measures to fit these two expectations.

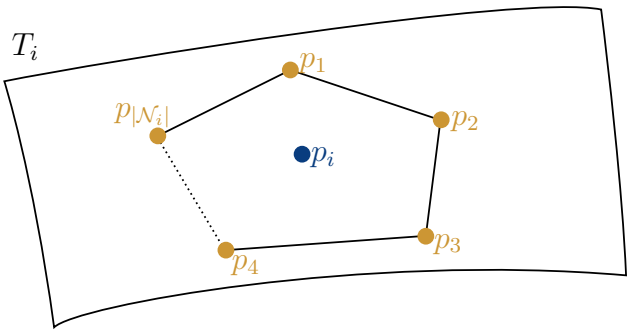


Figure 6.1: *Neighborhood of a point p_i on T_i given as a regular $|\mathcal{N}_i|$-gon.*

Partition of Unity Note that given these choices, our proposed weights around one point p_i do not satisfy partition of unity. This is, as the weights are to be used multiplicatively, see the applications in Section 6.3. In the base-case of a regular $|\mathcal{N}_i|$-gon—where uniformity of the sampling is given and nothing is to be changed—these multiplicative weights all have to be equal to 1. However, if partition of unity is necessary, the following normalizations (6.10), (6.12), and (6.15) can easily be altered by removing the term $|\mathcal{N}_i|$ from the numerator. The weights then immediately yield a partition of unity around the point p_i.

Based on this notation, in the following three Sections 6.2.1, 6.2.2, and 6.2.3 we will introduce three different approaches for weights to counteract non-uniformity in point sets. These can then each be used e.g. in the discretizations of operators as will be shown for the Laplace operator in Section 6.3.3.

6.2.1 Covariance Matrix Densities

In [LP05], the authors present an approach to estimate the density of a point set P in a given (tangential) direction from a point $p_i \in P$. They propose to use the eigenvalues and the eigenvectors of the covariance matrix built on a neighborhood $\mathcal{N}_i$ of p_i. Recall that the two eigenvectors $v_{i,1}, v_{i,2}$ of the covariance matrix corresponding to the larger two eigenvalues $\lambda_{i,1}, \lambda_{i,2}$ form an orthonormal basis of T_i. Therefore, any unit direction on T_i can be parametrized by $\varphi \in [0, 2\pi)$ as

$$e_\varphi = \cos(\varphi) v_{i,1} + \sin(\varphi) v_{i,2} \in T_i \tag{6.2}$$

with $\|e_\varphi\| = 1$. The density $\delta(e_\varphi)$ of P at p_i in direction e_φ is approximated using the quadratic form

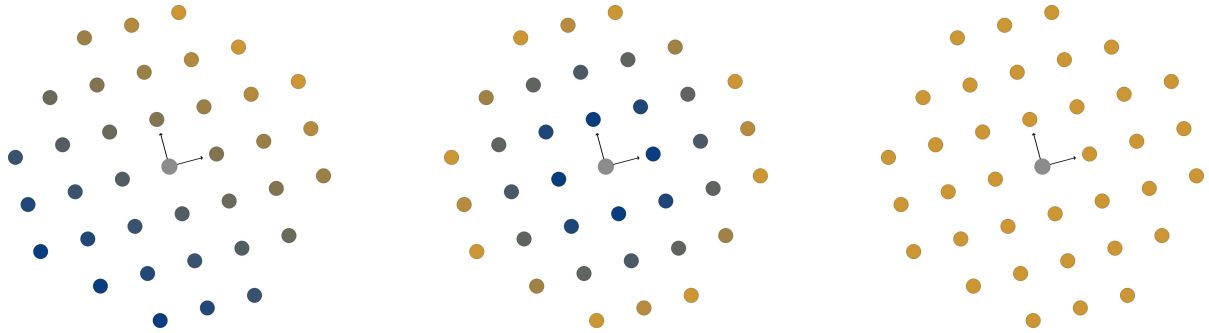
$$\delta(e_\varphi) = \delta_{i,1} \cos^2(\varphi) + \delta_{i,2} \sin^2(\varphi) \tag{6.3}$$

and the integral form of the tangential part of the diagonalized covariance matrix

$$\begin{pmatrix} \lambda_{i,1} & 0 \\ 0 & \lambda_{i,2} \end{pmatrix} = \frac{1}{2\pi} \int_0^{2\pi} \delta(e_\varphi) e_\varphi e_\varphi^t \, d\varphi. \tag{6.4}$$

In particular, we obtain $\delta(v_{i,1}) = \delta_{i,1}$ for direction $e_0 = v_{i,1}$ and $\delta(v_{i,2}) = \delta_{i,2}$ for direction $e_{\pi/2} = v_{i,2}$. Inserting the quadratic form (6.3) into the integral (6.4) and computing the integral component-by-component, $\delta_{i,1}$ and $\delta_{i,2}$ can be expressed in terms of the eigenvalues $\lambda_{i,1}$ and $\lambda_{i,2}$ by

$$\delta_{i,1} = 3\lambda_{i,1} - \lambda_{i,2}, \qquad \delta_{i,2} = 3\lambda_{i,2} - \lambda_{i,1}. \tag{6.5}$$



(a) *Density δ^{LP} given in (6.8) evaluated around point p_i on a uniform sampling. The density does not reflect the uniform sampling but assigns different weights to the neighbors.*

(b) *Density δ given in (6.6) evaluated around point p_i on a uniform sampling. Note how the different distances affect the density measure.*

(c) *Density $\bar{\delta}$ given in (6.7) evaluated around point p_i on a uniform sampling. Note how all points are assigned similar values.*

Figure 6.2: *Density measures δ^{LP} (6.8) and δ with both regular (6.6) and normalized input (6.7) on the same, uniformly sampled point set. Values of δ^{LP} and δ ranging from low (blue) to high (yellow). Note how δ^{LP} and δ as in (6.6) assign varying density values although the neighborhood is very uniformly sampled, while $\bar{\delta}$ as in (6.7) does assign equal density values.*

The corresponding calculations are given in Appendix D. Since the eigenvectors $v_{i,1}, v_{i,2}$ form an orthonormal basis of the tangent space, we have $\langle e_\varphi, v_{i,1}\rangle = \cos(\varphi)$ and $\langle e_\varphi, v_{i,2}\rangle = \sin(\varphi)$. Plugging these expressions and the equalities (6.5) into (6.3), we obtain the directed density measure

$$\delta(e_\varphi) = (3\lambda_{i,1} - \lambda_{i,2})\langle e_\varphi, v_{i,1}\rangle^2 + (3\lambda_{i,2} - \lambda_{i,1})\langle e_\varphi, v_{i,2}\rangle^2.$$

For a point $p_i \in P$, the density in direction of a neighbor $p_j \in \mathcal{N}_i$ can then be computed as

$$\delta(e_{ij}^{\tan}) = (3\lambda_{i,1} - \lambda_{i,2})\langle e_{ij}^{\tan}, v_{i,1}\rangle^2 + (3\lambda_{i,2} - \lambda_{i,1})\langle e_{ij}^{\tan}, v_{i,2}\rangle^2, \tag{6.6}$$

where $e_{ij}^{\tan}$ denotes the tangential part of the vector $e_{ij} = p_j - p_i$. Note that this expression is not strictly dependent on the direction, as the scalar products obtain different values with varying length of $e_{ij}^{\tan}$. Therefore, we normalize the argument and obtain

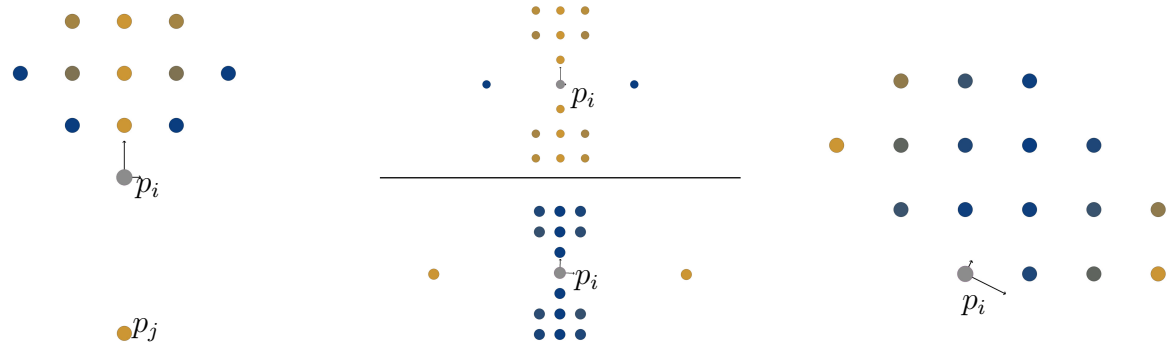
$$\bar{\delta}(e_{ij}^{\tan}) = (3\lambda_{i,1} - \lambda_{i,2})\langle e_{ij}^{\tan} / \left\|e_{ij}^{\tan}\right\|, v_{i,1}\rangle^2 + (3\lambda_{i,2} - \lambda_{i,1})\langle e_{ij}^{\tan} / \left\|e_{ij}^{\tan}\right\|, v_{i,2}\rangle^2. \tag{6.7}$$

In [LP05], the authors give the density measure slightly differently, namely as

$$\delta^{LP}(e_{ij}^{\tan}) = \frac{3\lambda_{i,1} - \lambda_{i,2}}{2}\langle e_{ij}^{\tan} / \left\|e_{ij}^{\tan}\right\|, v_{i,1}\rangle + \frac{3\lambda_{i,2} - \lambda_{i,1}}{2}\langle e_{ij}^{\tan} / \left\|e_{ij}^{\tan}\right\|, v_{i,2}\rangle. \tag{6.8}$$

This measure obtains both positive and negative values, which is not suitable to be used as weights. In particular, even for quite uniformly sampled neighborhoods, the points in the neighborhood are not attributed equal weights, see Figure 6.2a. Therefore, we assume that the authors of [LP05] meant to give (6.7) as density measure, which results from the above computations and which assigns equal weights to uniformly sampled neighborhoods, see Figure 6.2b for regular and Figure 6.2c for normalized input.

Note that expression (6.7) is symmetric with respect to its argument, that is, $\bar{\delta}(e_{ij}^{\tan}) = \bar{\delta}(-e_{ij}^{\tan})$. Therefore, a point $p_j \in \mathcal{N}_p$ in direction $e_{ij}^{\tan}$ that lies in a very sparsely sampled area might still



(a) *The point $p_j \in \mathcal{N}_i$ lies in a sparsely sampled area. However, the density measure $\bar{\delta}$ as given in (6.7) still assigns a high value to p_j as there is a densely sampled area on the other side of p_i, in direction $-e_{ij}$.*

(b) *Density $\bar{\delta}$ as given in (6.7) evaluated around point p_i on two slightly varying neighborhoods. While the upper neighborhood is assigned expected values, the assignment in the lower neighborhood is off because of slight variations.*

(c) *Density $\bar{\delta}$ given in (6.7) evaluated around point p_i not being close to the barycenter of the given neighborhood. Assigned density values are high on the sparse points to the ends of the neighborhood and low in the sole densely sampled area.*

Figure 6.3: *Problems of the density $\bar{\delta}$ given in (6.7) because of symmetry, outliers, and placement of the neighborhood. Densities computed from $\bar{\delta}$ ranging from low (blue) to high (yellow).*

get assigned high density weight if points on the opposite side $-e_{ij}^{\tan}$ of p_i are sampled densely, see Figure 6.3a. Furthermore, the density weight is sensitive to variations in the neighborhood. Moving one neighbor $p_j \in \mathcal{N}_i$ far away from p_i, the direction of the first principal component changes and thereby also the density measures assigned to the neighborhood. Consider the situation shown in Figure 6.3b, where two outlier drastically change the density values. Finally, if the point p_i is not located close to the barycenter of $\mathcal{N}_i$, the assigned weights are not necessarily accurate, as given in Figure 6.3c.

Adding to the shortcomings of $\bar{\delta}$ listed above, it could also evaluate to a negative value. Namely, if $\lambda_{i,1} > 3\lambda_{i,2}$ and $e_{ij}^{\tan}$ close to $v_{i,1}$, i.e. $\langle e_{ij}^{\tan}, v_{i,1}\rangle^2 \approx 1$, we get $\bar{\delta}(e_{ij}^{\tan}) < 0$. As the weights are required to be positive, the values of $\bar{\delta}$ have to be shifted to a strictly positive interval. We set

$$\delta^{\mathrm{Cov}}(e_{ij}^{\tan}) := \bar{\delta}(e_{ij}^{\tan}) + \max_{p_j \in \mathcal{N}_i} \{-\bar{\delta}(e_{ij}^{\tan}), 0\} \tag{6.9}$$

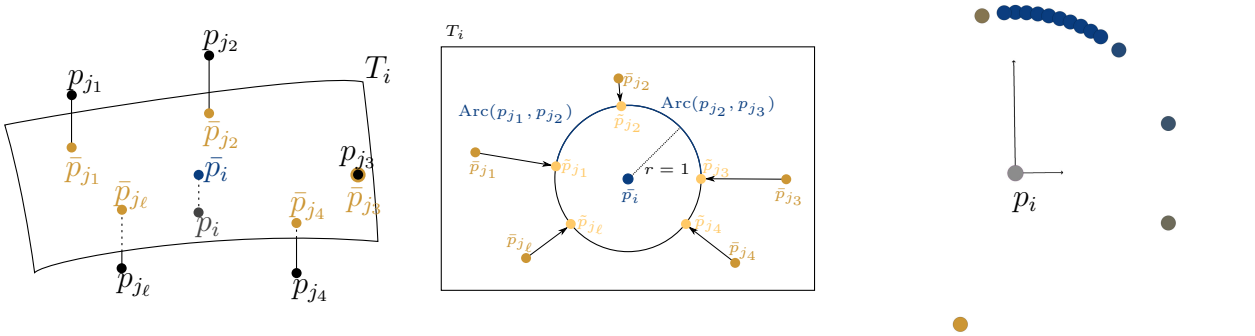
and obtain the first discrete density measure δ^{Cov}. Note that the adjustment of (6.9) is different for each $p_i \in P$. However, as the weights are to reflect the local density around p_i and not any global structure, these differences in the adjustment are not problematic, in particular after normalizing in (6.10).

In order to have a measure which gives large values for sparse directions and low values for dense directions, we reflect the values at their arithmetic mean

$$m_i^{\mathrm{Cov}} = \sum_{p_\ell \in \mathcal{N}_i} \delta^{\mathrm{Cov}}(e_{i\ell}^{\tan})/|\mathcal{N}_i|$$

and normalize to obtain a value between 0 and $|\mathcal{N}_i|$:

$$\sigma^{\mathrm{Cov}}(e_{ij}^{\tan}) = \frac{|\mathcal{N}_i|\left(2m_i^{\mathrm{Cov}} - \delta^{\mathrm{Cov}}(e_{ij}^{\tan})\right)}{\sum_{p_\ell \in \mathcal{N}_i}\left(2m_i^{\mathrm{Cov}} - \delta^{\mathrm{Cov}}(e_{i\ell}^{\tan})\right)} = 2 - \frac{\delta^{\mathrm{Cov}}(e_{ij}^{\tan})}{m_i^{\mathrm{Cov}}}. \tag{6.10}$$



(a) *Points $p_j \in \mathcal{N}_i$ and the center point p_i are projected to the tangent plane T_i.*

(b) *Points $\bar{p}_j$ are projected onto the unit circle around $\bar{p}_i$ on T_i. For each point p_j, its density is computed as half of the sum of the arc lengths on the circle starting at its projection $\tilde{p}_j$.*

(c) *Note how the leftmost point in the upper region gets a different weight than the other points. Values ranging from low density (blue) to high density (yellow).*

Figure 6.4: *Illustration of the arc length density given in (6.11). Points $p_j \in N_i$ are projected to the tangent plane T_i. There, they are projected to the circle of radius 1 around p_i. For each point p_j, its density is computed as half of the lengths of the arcs on the circle starting at its projection $\tilde{p}_j$. When applied to non-uniform dense samples, the boundary points of dense regions get ranked significantly different from the inner points of the region, as the arc to the neighboring sparse region is only distributed on the end points.*

In particular, if all weights are approximately equal, i.e. $\delta^{\text{Cov}}(e_{ij}^{\tan}) \sim m_i^{\text{Cov}}$ on a regular $|\mathcal{N}_i|$-gon, we have $\sigma^{\text{Cov}}(e_{ij}^{\tan}) \sim 1$ for all $p_j \in \mathcal{N}_i$.

6.2.2 Arc Length Density

In this section, we will derive a discrete density measure that does not suffer from the disadvantages of δ^{Cov} as listed above. It was proposed in [Jan17]. Given a point $p_i \in P$ and its neighborhood $\mathcal{N}_i$, we first project all neighbors $p_j \in \mathcal{N}_i$ as well as p_i to the tangent plane T_i and obtain their projections $\bar{p}_j, \bar{p}_i \in T_i$. The projected neighbors are then projected once more onto the circle of radius $r = 1$ around $\bar{p}_i$ on T_i, creating $\tilde{p}_j \in T_i$ with $\|\tilde{p}_j - \bar{p}_i\| = 1$ for all $p_j \in \mathcal{N}_i$, see Figures 6.4a and 6.4b.

Given an orientation of the tangent plane T_i by the normal at p_i, the points $\tilde{p}_j$ can be ordered along the unit circle around p_i by their angle φ_j. If $\varphi_j = \varphi_\ell$ for two points p_j, p_ℓ, we order by their indices j, ℓ. We set $\varphi(v_1) = 0$. Denote the order by $\tilde{p}_{j_1}, \dots, \tilde{p}_{j_{|\mathcal{N}_i|}}$. For any point $\tilde{p}_{j_\ell}$, $\ell \in \{1, \dots, |\mathcal{N}_i|\}$, consider the arc length on the unit circle from $\tilde{p}_{j_{\ell-1}}$ to $\tilde{p}_{j_\ell}$ and from $\tilde{p}_{j_\ell}$ to $\tilde{p}_{j_{\ell+1}}$, see Figure 6.4b.

The main idea for the density measure is now to assume a direction $e_{ij}^{\tan}$ to point into a dense area, if the arcs to the two neighbors of $\tilde{p}_j$ are short compared to the longest possible arc length 2π. The second discrete measure is then given by the sum of half the lengths of the adjacent arcs. For a point $p_j \in \mathcal{N}_i$ with projection $\tilde{p}_{j_\ell}$ we define

$$\delta^{\text{Arc}}(e_{ij}^{tan}) := \frac{\sphericalangle(\tilde{p}_{j_{\ell-1}} - \bar{p}_i, \tilde{p}_{j_\ell} - \bar{p}_i)}{2} + \frac{\sphericalangle(\tilde{p}_{j_\ell} - \bar{p}_i, \tilde{p}_{j_{\ell+1}} - \bar{p}_i)}{2} \tag{6.11}$$

with the angle $\sphericalangle$ between two vectors given in radians. Recall that the length a of an arc of angle $\phi \in [0, 2\pi]$ is given by $a = r \cdot \phi$, but since we project to the unit circle, this reduces to $a = \phi$.

Furthermore, δ^{Arc} measures the reciprocal of the density, as it assign small values to dense regions and large values to sparse regions.

Although this definition does not suffer from the problematic symmetry as δ^{Cov}, there is still a slight inconvenience. Consider a densely sampled region neighboring a sparsely sampled region as shown in Figure 6.4c. The circle arc separating the sparse point and the dense region is solely contributing to the density measure of the border point of the dense region, assigning it a significantly higher value than the other points of the dense region.

As in Section 6.2.1, we will now normalize the measure (6.11) in order to obtain positive weights that are about 1 on a regular $|\mathcal{N}_i|$-gon. Therefore, we set

$$\sigma^{\mathrm{Arc}}(e_{ij}^{\tan}) = \frac{\delta^{\mathrm{Arc}}(e_{ij}^{\tan})|\mathcal{N}_i|}{\sum_{p_k \in \mathcal{N}_i} \delta^{\mathrm{Arc}}(e_{ij}^{\tan})}. \tag{6.12}$$

As in (6.10), for equal weights in our test case of the regular $|\mathcal{N}_i|$-gon, we have $\sigma^{\mathrm{Arc}}(e_{ij}^{\tan}) \sim 1$ for all $p_j \in \mathcal{N}_i$.

6.2.3 Smooth Basis Density

Having presented two discrete density measures δ^{Cov} and δ^{Arc} above, we will now present a smooth density measure, based on the discrete data given. The first steps are the same as in Section 6.2.2. That is, we consider the projection $\tilde{p}_j$ of all points $p_j \in \mathcal{N}_i$ to a unit circle on T_i around p_i, see Figure 6.4. As before, each point $\tilde{p}_j$ is assigned an angle φ_j with $\varphi(v_1) = 0$. The main idea is now to define a basis function on each $\tilde{p}_j$ and thereby obtain a density measure at any angle $\varphi \in [0, 2\pi)$ by

$$\delta^{\mathrm{RBF}} : [0, 2\pi) \to \mathbb{R}_{\geq 0}, \qquad \delta^{\mathrm{RBF}}(\varphi) = \frac{1}{|\mathcal{N}_i|} \sum_{p_j \in \mathcal{N}_i} \psi_j(\varphi). \tag{6.13}$$

The basis function ψ_j should be of finite local support to only influence a small neighborhood around the sample point p_j. It should furthermore be smooth to obtain a smooth density measure δ^{RBF}. Finally, the size of the local support should be chosen such that uniformly distributed samples p_j lead to a preferably uniform density. Therefore, we propose the bump function as local basis

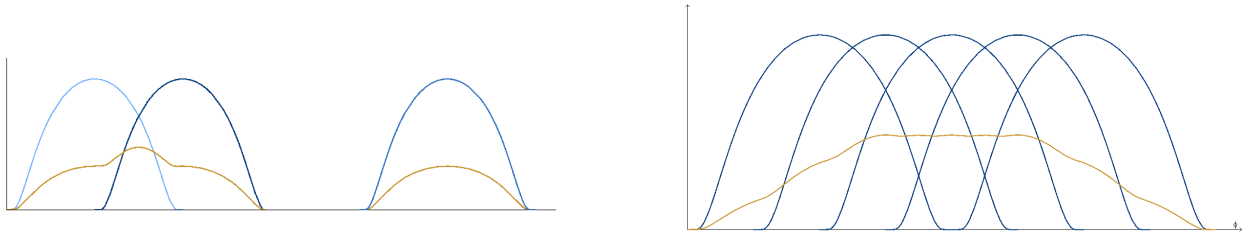
$$\psi_j : [0, 2\pi) \to \mathbb{R}_{\geq 0}, \qquad \psi_j(\varphi) = \begin{cases} \exp\left(\frac{1}{r_i^{-2}(\varphi-\varphi_j)^2 - 1}\right) & \varphi \in (-r_i, r_i), \\ 0 & \text{otherwise} \end{cases} \tag{6.14}$$

with $r_i = \frac{2\pi}{|\mathcal{N}_i|}$. Note that ψ_j is $\mathcal{C}^\infty$ and of finite local support. See Figure 6.5a for a corresponding illustration of equation (6.13) and see Figure 6.5b for a plot of the resulting density measure on a uniform point sample.

Once more, as in (6.10) and (6.12), we will normalize the values of δ^{RBF} to have density weights of approximately 1 on the regular $|\mathcal{N}_i|$-gon. Therefore, we set

$$\sigma^{\mathrm{RBF}}(e_{ij}^{\tan}) = \frac{\delta^{\mathrm{RBF}}(e_{ij}^{\tan})|\mathcal{N}_i|}{\sum_{p_k \in \mathcal{N}_i} \delta^{\mathrm{RBF}}(e_{ij}^{\tan})}. \tag{6.15}$$

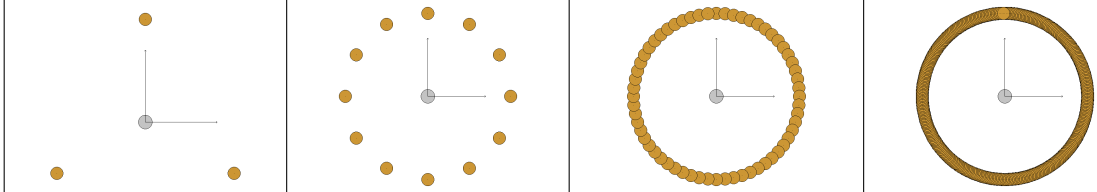
Concluding Overview In the Sections 6.2.1–6.2.3 we presented three different directional density measures on point sets. Namely, we built on a method by [LP05] to define a measure



(a) *Three basis functions (6.14) shown in blue, as well as the corresponding density δ^{RBF}, (6.13), shown in yellow.*

(b) *Several basis functions on a set of uniform samples shown in blue. Note how the corresponding density (shown in yellow) becomes almost uniform in the center.*

Figure 6.5: *Plotting several basis functions (6.14) in blue and the corresponding densities (6.13) in yellow.*



k	$k=3$	$k=12$	$k=60$	$k=360$
σ^{Cov}	$2 \cdot 10^{-16}$	$4 \cdot 10^{-16}$	$7 \cdot 10^{-16}$	$13 \cdot 10^{-16}$
σ^{Arc}	$1 \cdot 10^{-16}$	$7 \cdot 10^{-16}$	$197 \cdot 10^{-16}$	$7736 \cdot 10^{-16}$
σ^{RBF}	0.0	$1 \cdot 10^{-16}$	$4 \cdot 10^{-16}, -4 \cdot 10^{-16}$	$82 \cdot 10^{-16}$

Table 6.1: *Results of density measures σ^{Cov} (6.10), σ^{Arc} (6.12), and σ^{RBF} (6.15) on a regular k-gon neighborhood for varying k. The numbers indicate the maximum deviation of the weights from* 1.0.

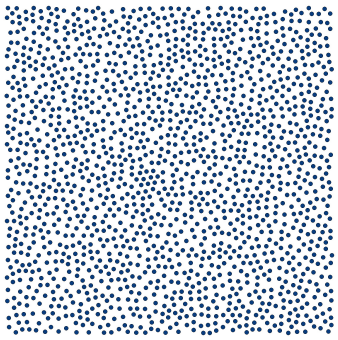
δ^{Cov}, (6.9), based on the covariance matrix. Furthermore, we projected points to an estimated tangent plane and distributed the arc lengths of the unit circle around the center point as density measure δ^{Arc}, (6.11). Finally, we defined smooth radial basis functions for each sample p_j and summed these up to obtain a smooth density measure δ^{RBF}, (6.15).

All three measures are to be used as weights in discretizations of differential geometry operators as described in the beginning of Section 6.2. Therefore, we normalized them to our test case: the regular $|\mathcal{N}_i|$-gon. We obtained three corresponding weights σ^{Cov}, (6.10), σ^{Arc}, (6.12), and σ^{RBF}, (6.15). In the next section, these different weights will be evaluated experimentally.

6.3 Experimental Results

In this section, we summarize experimental results proving the effectiveness of our approach. As a first preliminary test, we compute the different normalized density measures σ^{Cov} (6.10), σ^{Arc} (6.12), and σ^{RBF} (6.15) on k nearest neighborhoods consisting of regular k-gons. Thereby, we justify the definitions of the respective measures by showing that they obtain weights around 1 as desired. In Table 6.1, we presented the computed maximal deviation from weight 1 for the three measures and different k. Note that all weights lie well within a range around 1. We find that the values of δ^{Arc} seem to deviate most. However, this comes with growing values of k, where in any real application typically small values of k are used.

Following a suggestion of Scott Schaefer, we also evaluate our weights on a blue noise sample.



200 Blue Noise Experiments	σ^{Cov}	σ^{Arc}	σ^{RBF}
Average	0.200	0.239	0.258
Minimum	0.000	0.000	0.000
Maximum	1.000	1.560	1.026
Standard Deviation	0.092	0.054	0.042

Figure 6.6: *A blue noise sample on a* 500×500 *domain computed following [Bri07]. To the right the results of the density measures* σ^{Cov} *(6.10),* σ^{Arc} *(6.12), and* σ^{RBF} *(6.15) on* 200 *blue noise samples. The numbers indicate the respective deviation of the weights from* 1.0*. Average, minimum, maximum, and standard deviation are taken over all weights* $\sigma(e_{ij}^{\tan})$ *computed on all pairs of neighboring points* p_i, p_j *in all* 200 *experiments.*

In a flat domain, this type of noise is given by a 2D array of scalar values arranged such that the Fourier power spectrum of any thresholded gray-level is isotropic and devoid of low frequencies, see [GF16]. For our implementation we follow an approach by Bridson[1], see [Bri07]. Our domain is of size 500×500 with 30 tries to create a new sample from any active sample and radius 10. A resulting sample is shown in Figure 6.6. We create 200 such samples, in each, we compute the densities σ^{Cov}, σ^{Arc}, and σ^{RBF} for each point lying within coordinates $[100, 400] \times [100, 400]$ to exclude problematic areas at the boundary. Thereby, in our experiments, on average 567.185 points were considered. As we expect to achieve a density close to 1, for each point p_i with neighbors p_j, we consider the deviation from 1. Results of the experiments are also given in Figure 6.6. Note that on average, the measure σ^{Cov} has least deviation from the ideal weights, but its maximum deviation is comparable to that of σ^{RBF} while its standard Deviation is more than twice as large.

6.3.1 Discrete Shape Operator

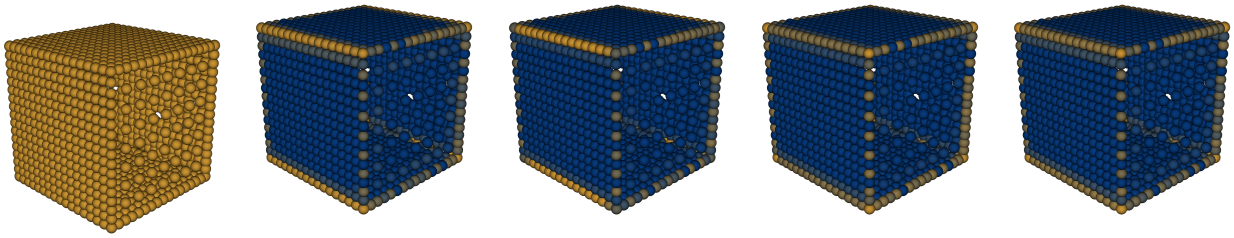
Having passed these first preliminary experiments, we will benchmark our weights on a discretization of the shape operator originally proposed by Taubin in [Tau95]. The discretization has been used on point sets with some alterations discussed in [LP05; Skr14b; Jan17]. The main idea is to discretize a matrix M_i at point $p_i \in P$ as

$$M_i = \frac{1}{2\pi} \sum_{p_j \in \mathcal{N}_i} \omega_{ij} \kappa_{ij} e_{ij}^{\tan} e_{ij}^{\tan t}, \tag{6.16}$$

where $e_{ij}^{\tan}$ is the same vector as in Section 6.2, ω_{ij} is a weight assigned to each point in order to have a faithful approximation, and κ_{ij} is the directional curvature for the direction $e_{ij}^{\tan}$. It is discretized as

$$\kappa_{ij} = \frac{2\langle n_i, e_{ij} \rangle}{\|e_{ij}\|},$$

[1]The implementation used can be found here: https://stackoverflow.com/questions/32979413/infinite-blue-noise



(a) *input point set (1186 samples)* **(b)** *max. principal curvature with* ω_{ij}^{Tau} *(6.17)* **(c)** *max. principal curvature with* $\omega_{ij}^{\mathrm{Cov}}$ *(6.18)* **(d)** *max. principal curvature with* $\omega_{ij}^{\mathrm{Arc}}$ *(6.18)* **(e)** *max. principal curvature with* $\omega_{ij}^{\mathrm{RBF}}$ *(6.18)*

Figure 6.7: *Plotting maximum principal curvature on a cube with differently sampled sides for weights* ω_{ij} *as indicated ranging from the respectively lowest value (blue) to the respectively highest value (yellow).*

where n_i is the normal at p_i. With this setup, the principal curvatures $\kappa_{i,1}$, $\kappa_{i,2}$ at point $p_i \in P$ can be computed from the eigenvalues $m_{i,1}, m_{i,2}$ of M_i by

$$\kappa_{i,1} = 3m_{i,1} - m_{i,2}, \qquad \kappa_{i,2} = 3m_{i,2} - m_{i,1}.$$

For the weights ω_{ij}, in his article [Tau95], Taubin suggests to use

$$\omega_{ij}^{\mathrm{Tau}} = \frac{2\pi \left\| e_{ij} \right\|}{\sum_{p_j \in \mathcal{N}_i} \left\| e_{ij} \right\|}, \tag{6.17}$$

which we will compare to our, properly adjusted weights:

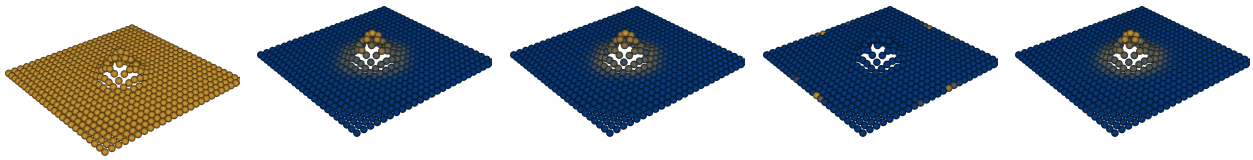
$$\omega_{ij}^{\mathrm{Cov}} = \frac{2\pi\sigma_{ij}^{\mathrm{Cov}} \left\| e_{ij} \right\|}{\sum_{p_j \in \mathcal{N}_i} \left\| e_{ij} \right\|}, \qquad \omega_{ij}^{\mathrm{Arc}} = \frac{2\pi\sigma_{ij}^{\mathrm{Arc}} \left\| e_{ij} \right\|}{\sum_{p_j \in \mathcal{N}_i} \left\| e_{ij} \right\|}, \qquad \omega_{ij}^{\mathrm{RBF}} = \frac{2\pi\sigma_{ij}^{\mathrm{RBF}} \left\| e_{ij} \right\|}{\sum_{p_j \in \mathcal{N}_i} \left\| e_{ij} \right\|}. \tag{6.18}$$

Synthetic Models We will first run some tests on synthetic models before testing the weights on real world models. A first test case is a cube on which two opposing sides are sampled with only a quarter of the points compared to each of the other sides, see Figure 6.7a. On this point set, we test the different weights and plot the maximum principal curvature $\max(\kappa_{i,1}, \kappa_{i,2})$ for each point $p_i \in P$. Images are created with $k = 12$ and relaxed neighborhoods as defined in Equation (6.1).

Note how the weights $\omega_{ij}^{\mathrm{Tau}}$ and $\omega_{ij}^{\mathrm{Cov}}$ assign higher values to the edge between two equally dense sampled sides than to the edges bridging two differently sampled sides of the cube. The assigned curvature values on the edges are more regular for weights $\omega_{ij}^{\mathrm{Arc}}$ and $\omega_{ij}^{\mathrm{RBF}}$, also the corners are marked as points of high principal curvature.

In Section 6.2.1, we saw that the weights $\omega_{ij}^{\mathrm{Cov}}$ exhibit a problematic symmetrical behavior. This effect leads to an overestimate of those edges of the cube neighboring two dense regions. Furthermore, it prevents the weights $\omega_{ij}^{\mathrm{Cov}}$ to determine the corners of the cube as points of highest curvature.

A second test case is a Gaussian bump on an otherwise flat plane, see Figure 6.8a, on which we test the different weights and plot the maximum principal curvature $\max(\kappa_{i,1}, \kappa_{i,2})$ for each point $p_i \in P$. Images are created with $k = 12$ and relaxed neighborhoods as defined in Equation (6.1). Note that in this case, weights $\omega_{ij}^{\mathrm{Tau}}$, $\omega_{ij}^{\mathrm{Cov}}$, and $\omega_{ij}^{\mathrm{RBF}}$ perform very similar. However, the weights



(a) *input point set (1186 samples)* **(b)** *max. principal curvature with* ω_{ij}^{Tau} *(6.17)* **(c)** *max. principal curvature with* $\omega_{ij}^{\mathrm{Cov}}$ *(6.18)* **(d)** *max. principal curvature with* $\omega_{ij}^{\mathrm{Arc}}$ *(6.18)* **(e)** *max. principal curvature with* $\omega_{ij}^{\mathrm{RBF}}$ *(6.18)*

Figure 6.8: *Plotting maximum principal curvature on a Gaussian bump on the plane for weights* ω_{ij} *as indicated ranging from the respectively lowest value (blue) to the respectively highest value (yellow).*

$\omega_{ij}^{\mathrm{Arc}}$ are not able to recover the curvature of the bump. That is, because weights for some points on the border of the geometry become very large, as the arcs reach almost a length of π there. These throw off the curvature estimates for the whole geometry.

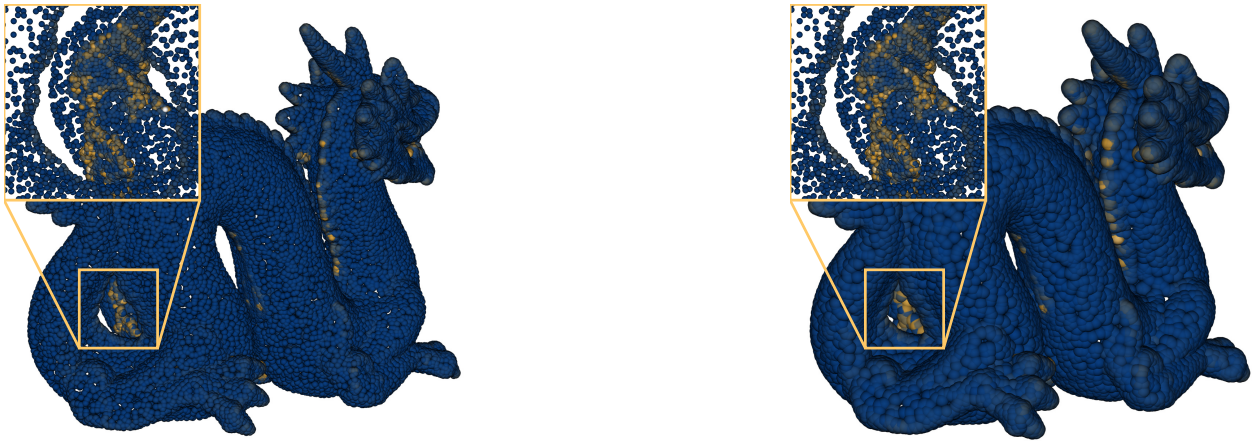
A third and final test on a synthetic model is run on an octahedron with tangential noise added to it. The test is run only on the so far supreme weights $\omega_{ij}^{\mathrm{tau}}$ and $\omega_{ij}^{\mathrm{RBF}}$. Figure 6.9a shows how despite the noise, weights $\omega_{ij}^{\mathrm{RBF}}$ emphasize the edges and corners of the octahedron better than weights $\omega_{ij}^{\mathrm{tau}}$.

CAD and Real World Models Due to the shortcomings found for $\omega_{ij}^{\mathrm{Cov}}$ and $\omega_{ij}^{\mathrm{Arc}}$ in the previous investigation on synthetic models, in the following, we will concentrate on weights $\omega_{ij}^{\mathrm{Tau}}$ and $\omega_{ij}^{\mathrm{RBF}}$. We will test the two weights on several CAD and real world models. For each, we evaluate the range of maximum principal curvatures detected and also give a visual evaluation of certain interesting features.

In Figures 6.9 and 6.10, we visually compare the results of principal maximum curvature visualization on six CAD and real world geometries. All experiments were performed with $k = 12$ neighbors and relaxed neighborhoods as defined in (6.1). Note that the curvature assignments with weights $\omega_{ij}^{\mathrm{RBF}}$ emphasize features better, as highlighted in Figures 6.9b and 6.10a. Also, the values of curvature are more regular in particular along sharp features, even if curved, as highlighted in Figure 6.10a. The effect of noise on the curvature computation is shown in Figures 6.10b and 6.10c. In both cases, weights $\omega_{ij}^{\mathrm{RBF}}$ are slightly more resilient in presence of noise and emphasize features better. In particular, all examples show that utilizing the additional weights, no artifacts are introduced and no curvature estimates become flawed.

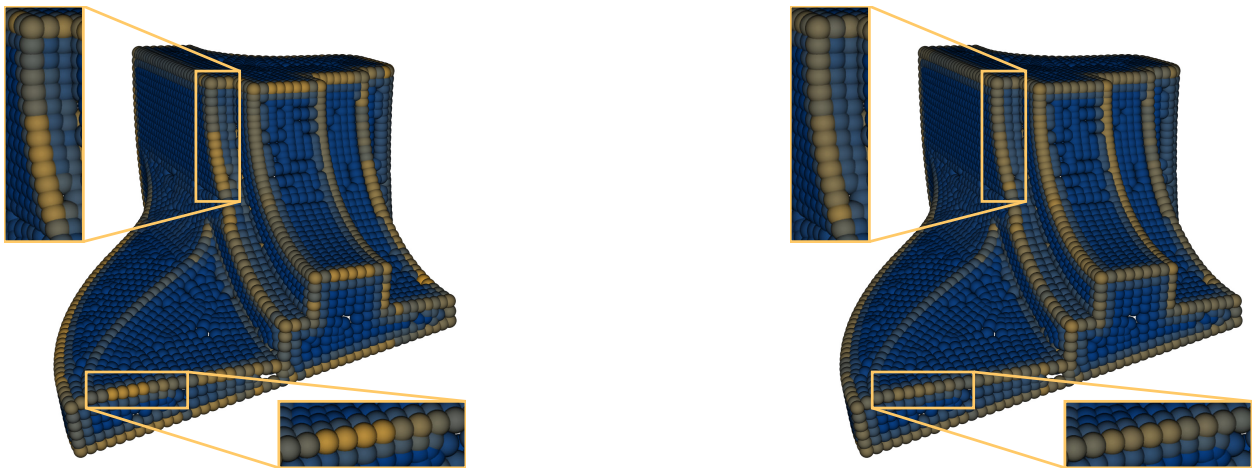
(a) *Maximum principal curvature on a noisy octahedron with weights ω_{ij}^{tau} (6.17) on the left and $\omega_{ij}^{\mathrm{RBF}}$ (6.18) on the right. Note how despite the noise, weights $\omega_{ij}^{\mathrm{RBF}}$ still capture the edges of the octahedron uniformly and the corners more precisely than weights ω_{ij}^{tau}.*

(b) *Maximum principal curvature of the bearing model with weights ω_{ij}^{tau} (6.17) on the left and $\omega_{ij}^{\mathrm{RBF}}$ (6.18) on the right. Note how the weights $\omega_{ij}^{\mathrm{RBF}}$ recover more points of low curvature in the area highlighted in the lower left and assign higher curvature to the tightly curved area highlighted in the upper right.*

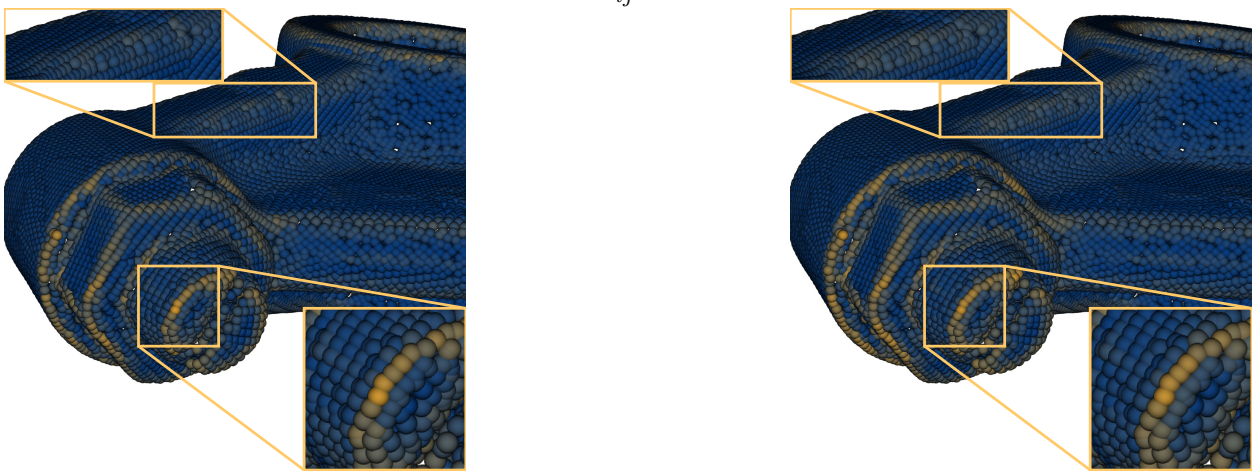


(c) *Maximum principal curvature of the dragon model with weights ω_{ij}^{tau} (6.17) on the left and $\omega_{ij}^{\mathrm{RBF}}$ (6.18) on the right. Note how independent of the weights, large clusters of points still get assigned high curvature values.*

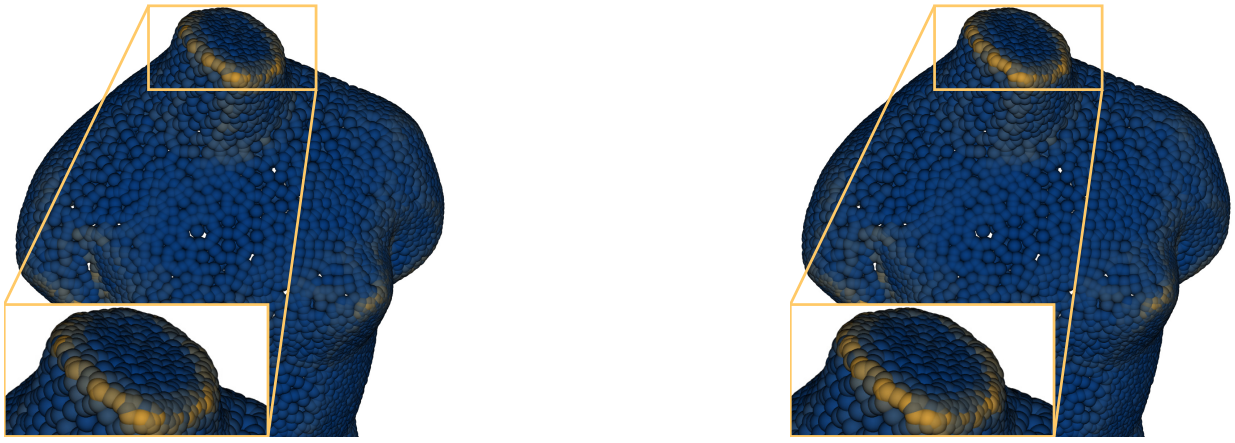
Figure 6.9: *Comparison of weights ω_{ij}^{tau} (6.17) and $\omega_{ij}^{\mathrm{RBF}}$ (6.18). Maximum principal curvatures are colored from the respective lowest value (blue) to the respective highest value (yellow).*



(a) *Maximum principal curvature of the dragon model with weights ω_{ij}^{tau} (6.17) on the left and $\omega_{ij}^{\mathrm{RBF}}$ (6.18) on the right. Note how the weights ω_{ij}^{tau} create a jump in the curvature assignments on both straight and bend edges. This jump is less present when using weights $\omega_{ij}^{\mathrm{RBF}}$.*



(b) *A mesh of the rocker arm model colored by maximum principal curvature with weights ω_{ij}^{tau} (6.17) on the left and weights $\omega_{ij}^{\mathrm{RBF}}$ (6.18) on the right.*



(c) *Maximum principal curvature of the Venus model with weights ω_{ij}^{tau} (6.17) on the left and $\omega_{ij}^{\mathrm{RBF}}$ (6.18) on the right.*

Figure 6.10: *Further comparison of weights ω_{ij}^{tau} (6.17) and $\omega_{ij}^{\mathrm{RBF}}$ (6.18). Note how the results are similar due to mostly good sampling of the models. Still, regions of curvature are covered better using weights $\omega_{ij}^{\mathrm{RBF}}$. Maximum principal curvatures are colored from the respective lowest value (blue) to the respective highest value (yellow).*

Model	#Points	**MPC** ω_{ij}^{Tau}	**MPC** ω_{ij}^{RBF}
Bearing	3,475	59.251	62.392
Dragon	50,000	2.551	2.363
Fandisk	6,475	14.142	15.014
noisy Rocker Arm	40,177	2.493	2.402
noisy Venus	17,018	22.539	23.560

Table 6.2: *Models and range of the maximum principal curvature (MPC) detected for the two weights* ω_{ij}^{Tau} *(6.17) and* ω_{ij}^{RBF} *(6.18). A higher range shows better curvature detection.*

Given that the distribution of points on the curvature range is comparable, a larger detected range means for higher sensitivity to the curvature of the geometry. The detected highest maximum principal curvature values for the tested models are given in Table 6.2. Note that all models include points of maximum principal curvature $\max\{\kappa_{i,1}, \kappa_{i,2}\} = 0$. Therefore, the values in Table 6.2 give the range of maximum principal curvature for the models. The assumption at this point is that higher curvature range means for a more subtle detection of differences in curvature, as the different features detected are compared along a larger range. That is, a higher maximum principal curvature allows to detect more nuances in the features of the model. Note that to this end the curvature range of ω_{ij}^{RBF} is higher or comparable to that of ω_{ij}^{Tau} (6.17) in all five models.

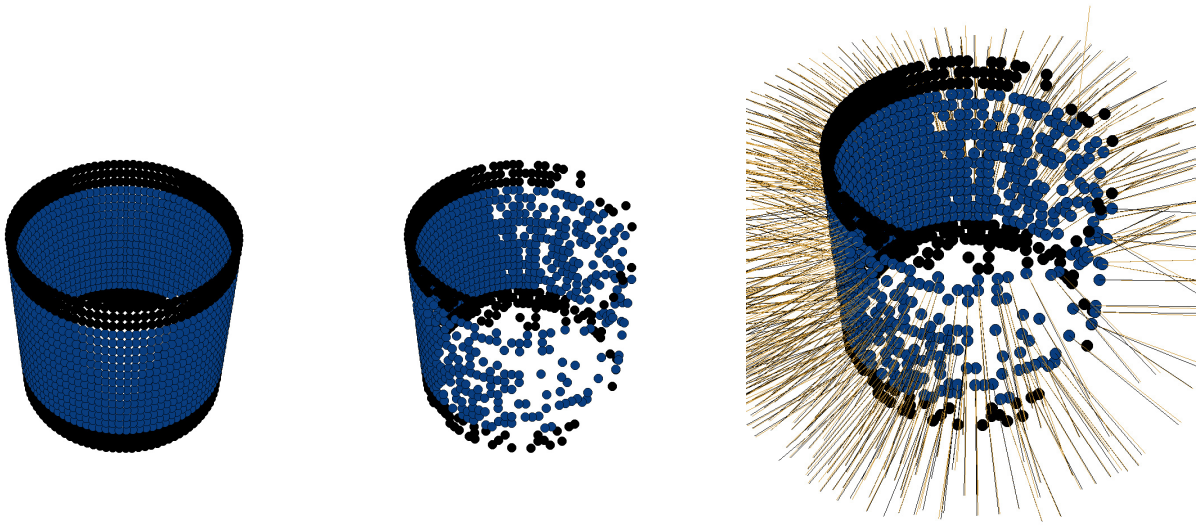
6.3.2 Normal Estimates

We proceed with our experiments by evaluating the proposed weights in the context of normal vector estimates. In order to have quantifiable results, we estimate the normals on a discretized cylinder body, see Figure 6.11a, where the actual normals can be computed easily and can be used as ground truth. As we are using the cylinder, the geometry has two boundary components. Since we aim for computation on surfaces, we run two sets of experiments: one, where we include the boundary and a second one, where we exclude the boundary by disregarding the lower and upper three sampling lines, marked black in Figure 6.11a.

We start with a cylinder discretized by 20 circles of radius 1 with 100 equally distributed points each. The circles lie 0.1 units apart. In order to evaluate on a non-uniform geometry, we delete points from this sampling utilizing the following scheme: For each point p_i, we draw a number $\rho_i \in [0, 1]$ with uniform distribution. Then $p_i = (x_i, y_i, z_i)^T$ is removed from the sampling, if $\rho_i \geq (x_i + 1)/2 + 0.05$. Thereby, we create a non-uniformly sampled cylinder, see Figure 6.11b, with $1,093$ points on average.

We run the experiment as follows: The true normal n_i at a point $p_i = (x_i, y_i, z_i)^T$ can easily be computed as $n_i = (x_i, y_i, 0)^T$. Note that this implies that the normals at the boundary are simply the limit of the normals from the cylinder surface in those cases of Table 6.3 where normals are included. This normal will serve as ground truth. For comparison, we compute the normals via principal component analysis (PCA). Here, we build the covariance matrix over a relaxed neighborhood ($k = 12$) of the considered point and utilize the eigenvector corresponding to the smallest eigenvalue as normal estimate n_i^{PCA}, see Figure 6.11c. We compare these with the results of a geographically weighted PCA, see [HBC11]. The weights are given by ω_{ij}^{Cov}, ω_{ij}^{Arc}, and ω_{ij}^{RBF} respectively. The respective normal n_i^{Cov}, n_i^{Arc}, n_i^{RBF} is then again given as eigenvector to the smallest eigenvalue of the resulting weighted covariance matrix.

The experiment was run $1,000$ times. All errors are given in degree ranging from 0 to 90. That is, e.g. the error of the PCA normal at point p_i is computed as $\measuredangle(n_i, n_i^{\text{PCA}}) \in [0, 90]$. The average error over all points and all experiments as well as the maximum error over all points and all



(a) *Original sample of cylinder on* $2,000$ *points.*

(b) *Cylinder after probabilistic thinning,* $1,093$ *points on average.*

(c) *Thinned cylinder with estimated normals from PCA (yellow) and ground truth normals (black).*

Figure 6.11: *Setup for the comparison of normal deviation.*

experiments is given in Table 6.3, as well as the standard deviation of the error over all points in all experiments. Note that all weighted normal estimates hit the maximal possible angle deviation from the ground truth normal. This is due to a faulty estimate of the normal in particular at the boundary or at points close to the boundary with badly sampled neighborhoods. For example, Figure 6.11c shows a point in the upper right that achieves maximum angle deviation. Especially the estimates with weights ω_{ij}^{Arc} are frequently off at the boundaries of the cylinder model which also effects the comparably high average angle deviation. On average, the normal estimates with weights ω_{ij}^{RBF} are about 2% better than the standard PCA estimates, in particular when including the problematic boundary.

		$\measuredangle(n_i, n_i^{\text{PCA}})$	$\measuredangle(n_i, n_i^{\text{Cov}})$	$\measuredangle(n_i, n_i^{\text{Arc}})$	$\measuredangle(n_i, n_i^{\text{RBF}})$
Including Boundary	Avg.	1.225	1.455	4.125	1.206
	Max.	71.386	90.000	90.000	90.000
	S.D.	1.350	4.482	15.929	1.655
Excluding Boundary	Avg.	0.031	0.033	0.079	0.031
	Max.	18.777	90.000	90.000	90.000
	S.D.	1.552	2.154	12.463	1.931

Table 6.3: *Average and maximum angle deviation as well as standard deviation of normals on a thinned out cylinder (Fig. 6.11b) and normal estimates with different weights.*

To further distinguish between the experimental values, we plot the average angle deviation obtained from PCA, ω_{ij}^{Cov}, and ω_{ij}^{RBF}, see Figure 6.12. Here, the experiments are sorted such that the average angle deviation of the PCA estimate grows from left to right. We find that the normals estimated with weights ω_{ij}^{RBF} are superior to those given by PCA estimate in 85.5% of all experiments. In the remaining 14.5%, the PCA normals were best. Neither weights ω_{ij}^{Cov} nor ω_{ij}^{Arc} achieved a lowest deviation in any of the experiments run. The maximal improvement of

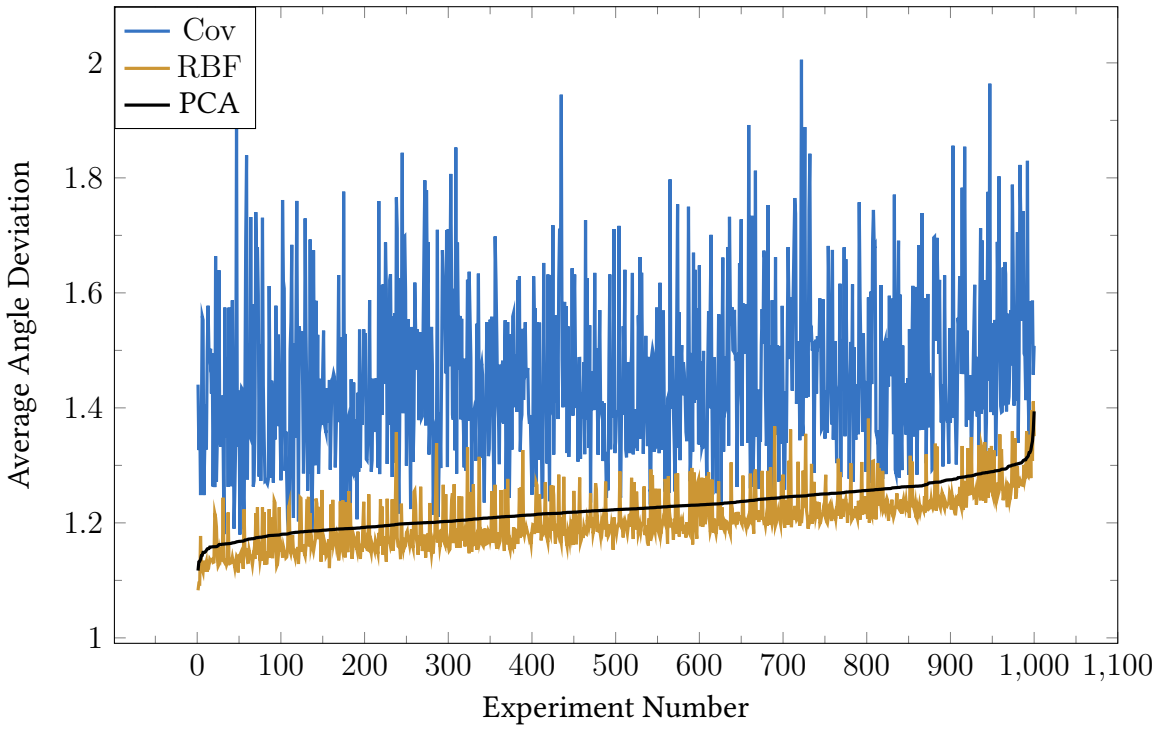


Figure 6.12: *Plotting the average angle deviation to the ground truth normal on the cylinder model including boundary (Fig. 6.11b) for* $1,000$ *experiments. The experiments are sorted such that the error of the PCA normal estimate grows from left to right. Note that the normal estimates by the weights* $\omega_{ij}^{\mathrm{RBF}}$ *lie below in* 85.5% *of all experiments.*

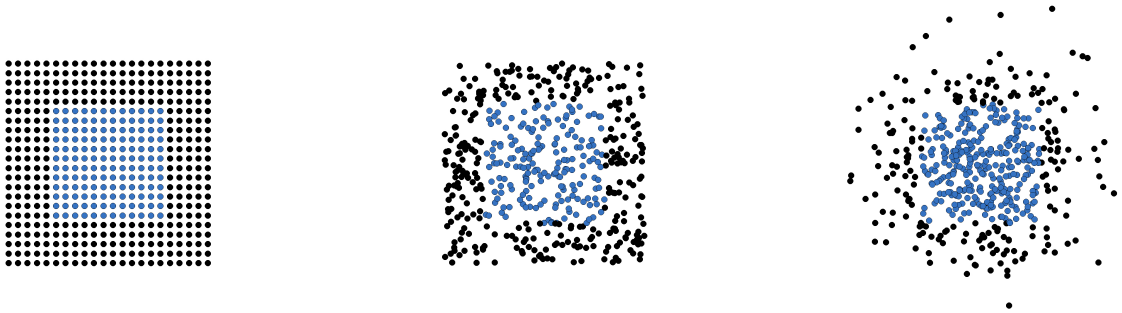
an estimate created with $\omega_{ij}^{\mathrm{RBF}}$ over a PCA normal was 0.069, that is 5.676 percent of the error made by PCA.

6.3.3 Laplace Operator

We finish this section on experimental results by evaluating the proposed weights in computations of the Laplace operator. As above, in order to have quantifiable results, we compute the Laplace operator for points sampling a flat two-dimensional square of length 1. For a uniform sampling, the Laplace operator should be $\vec{0}$ at all points except for boundary points. Therefore, we only take points $p_i = (x_i, y_i)^T$ into account where $0.2 \leq x_i, y_i \leq 0.8$. All experiments are run with relaxed neighborhoods where $k = 12$. Thus, for all these points, $\vec{0}$ is the ground truth Laplacian.

We sample the square in three different ways (see Figure 6.13). First, we impose a uniform grid with 484 points on the square from which 144 are taken into account, see Figure 6.13a. Second, we add 500 points $p_i = (x_i, y_i)$ randomly, where $x_i, y_i \in [0, 1]$ are chosen uniformly at random, see Figure 6.13b. Here, on average 179.9 points are taken into account. Third and finally, we add 500 points randomly, where x_i, y_i are chosen from a Gaussian distribution with mean 0.5 and standard deviation 0.25, see Figure 6.13c. Here, on average 296.6 points are taken into account.

We run the experiment as follows: The true Laplace operator on any inner point of the square should be $\vec{0}$, as there is no deviation of the points in height direction. All computed Laplacians will thus be compared to the zero vector. For comparison, we compute the Laplace Operator



(a) *Regular grid sample of a unit square with* 484 *points from which* 144 *are taken into account.*

(b) *Random uniform sample of a unit square with* 500 *points from which on average* 179.9 *are taken into account.*

(c) *Random Gaussian sample of a unit square with* 500 *points from which on average* 296.6 *are taken into account.*

Figure 6.13: *Examples of point sets for the Laplace computations. The black points are simply auxiliary where the blue points are those taken into account.*

utilizing the graph Laplacian, correspondingly weighted. That is, given a point p_i, we compute

$$\triangle p_i = \sum_{j \in \mathcal{N}_i} \omega_{ij} \cdot (p_j - p_i).$$

In case of the simple graph Laplacian we chose $\omega_{ij} = 1$ for all i, j. Otherwise, we use $\omega_{ij}^{\mathrm{Cov}}$, $\omega_{ij}^{\mathrm{Arc}}$, or $\omega_{ij}^{\mathrm{RBF}}$.

The experiment was run $1,000$ times on uniformly and Gaussian random data respectively. All errors are given as euclidean length of the computed vectors $\|\triangle p_i\|$. As expected, running computations on the regular grid (Figure 6.13a) yields values very close to zero independent of the used weights. Despite from this sanity check, the average error over all points and all experiments as well as the maximum error over all points and all experiments is given in Table 6.4. Furthermore, the table shows the standard deviation of the error over all points in all experiments. Note that the weights $\omega_{ij}^{\mathrm{Arc}}$ are highly instable in this setup and create numbers off the scale. In both experimental setups, the computation of the Laplace operator is about 4% better with weights $\omega_{ij}^{\mathrm{RBF}}$ when compared with the standard graph Laplacian.

		$\|\triangle p_i\|$	$\|\triangle^{\mathrm{Cov}} p_i\|$	$\|\triangle^{\mathrm{Arc}} p_i\|$	$\|\triangle^{\mathrm{RBF}} p_i\|$
Uniform	Average	0.311	0.343	$1.5 \cdot 10^{12}$	0.154
	Maximum	0.845	0.960	$2.0 \cdot 10^{16}$	0.788
	Standard Deviation	0.166	0.194	NaN	0.137
Gaussian	Average	0.161	0.177	NaN	0.125
	Maximum	0.982	1.061	$1.9 \cdot 10^{16}$	0.932
	Standard Deviation	0.165	0.191	$9.034 \cdot 10^{13}$	0.136

Table 6.4: *Average and maximum length of the computed Laplacian—i.e. deviation from the ground truth* $\vec{0}$*—as well as standard deviation.*

To further distinguish between the experimental values, we plot the average length of the Laplace operator obtained from the graph Laplacian, $\omega_{ij}^{\mathrm{Cov}}$, and $\omega_{ij}^{\mathrm{RBF}}$, see Figure 6.14. Here, the experiments are sorted such that the average vector length of the graph Laplacian grows from left

to right. We find that the Laplacians estimated with weights $\omega_{ij}^{\mathrm{RBF}}$ are superior to those given by graph Laplacian estimate in all experiments. Despite the high numerical error of the weights $\omega_{ij}^{\mathrm{Arc}}$ as shown in Table 6.4, they perform best in 59.3% of all run Laplace computations with uniform sampling and in case of Gaussian samples they perform best in 12.3% of all experiments. In all other cases, the best estimate is given by weights $\omega_{ij}^{\mathrm{RBF}}$. Tis hints at certain numerical problems in the computation of $\omega_{ij}^{\mathrm{Arc}}$ which are to be investigated in future research.

6.4 Conclusion

In this section, we presented three possible local directional density measures. We find that the density measure of [LP05] suffers from several shortcomings, therefore we introduce a different discrete measure and an additional smooth density measure. After deriving the measures theoretically, we prove their effectiveness on both synthetic and real world data in the context of maximum principal curvature computation, normal estimation, and the computation of the Laplace operator.

As in particular normals, but also the Laplace or the shape operator, are at the heart of all state-of-the-art applications involving point sets, the proposed weights are destined to have an impact on all these applications. Even though the effect is not always large, in a setup with a very high number of iterations, even a small improvement can make for a significant amount of time saved or accuracy gained. Therefore, the proposed weights are less of a self-contained tool, but rather an enhancement of currently available algorithms.

Different other applications—like the implementation of our weights into the contexts of point set registration, simplification, or surface reconstruction—are left for further research.

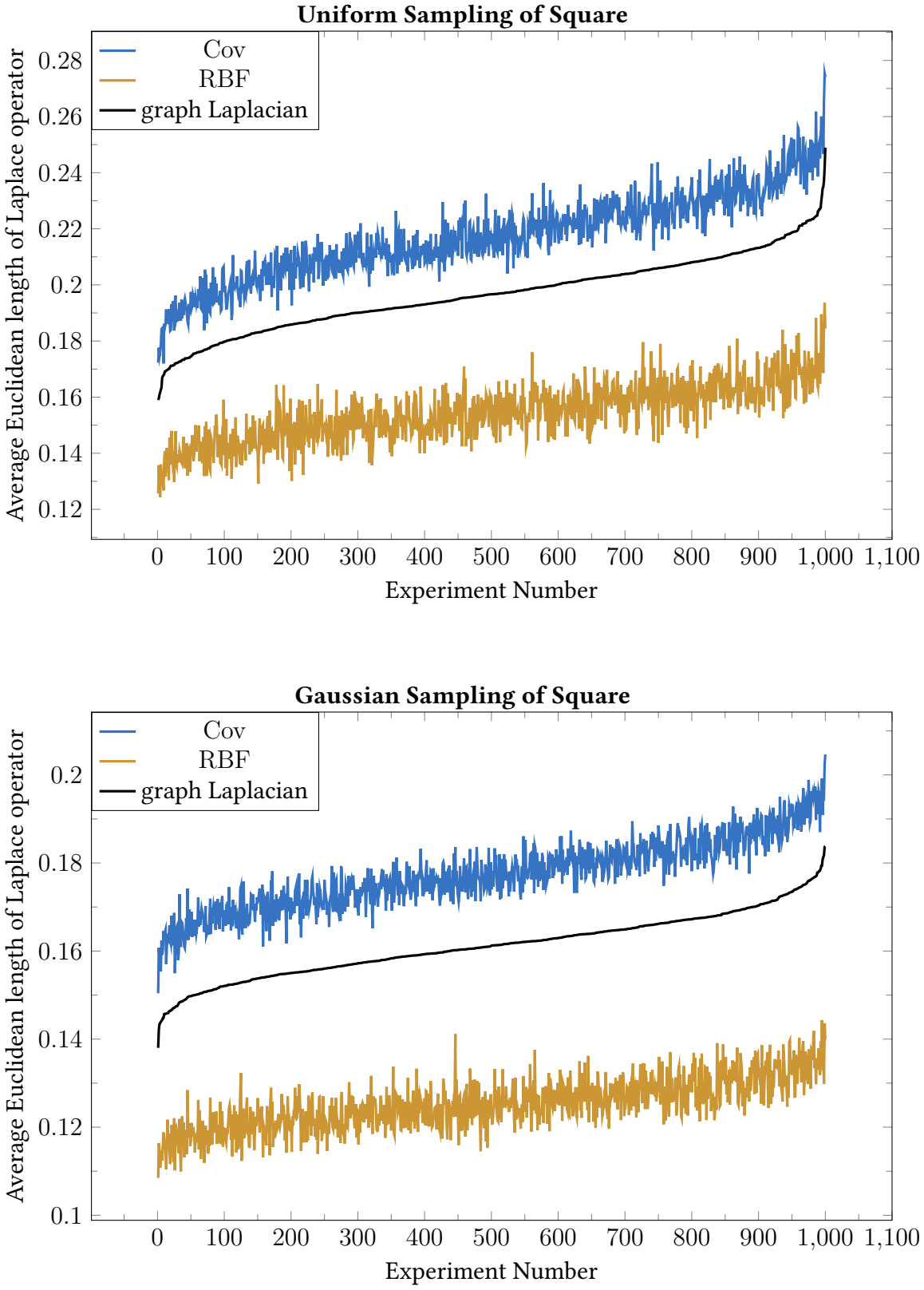


Figure 6.14: *Plotting the average Euclidean length of the Laplace operator on different samples of the unit square for* $1,000$ *experiments. The experiments are sorted such that the length of the graph Laplacian grows from left to right. Smaller values are better, as the Laplacian on the square should be* $\vec{0}$*. Note that the estimates with weights* $\omega_{ij}^{\mathrm{RBF}}$ *are always better than those derived by the other two methods.*

7 Feature Detection from Moving Least Squares

Many available algorithms tackling several tasks on point sets—like surface reconstruction or denoising—depend on the detection of features present in the originally sampled geometry. If these are not identified correctly they might be smoothed and thereby lost while denoising or they cannot be faithfully rebuilt from the point set when reconstructing the surface.

The purpose of this section is to present a feature recognition approach that is based in the framework of the Moving Least Squares (MLS) procedure, see Section 4.4.1. Therefore, it benefits from the mathematical guarantee (Theorem 13) given by MLS. The main contributions of this section are:

- Presentation of different approaches that use the mathematically proven functionality of the MLS technique to detect features.
- Experimental results on the feature detection sensitivity of the presented approaches.

7.1 Related Work

Feature detection is an area of active research in the context of polygonal meshes as well as for point sampled surfaces. A general overview of different techniques and an approach to feature detection via the Gauss Map is given in [WHH11]. However, only few of the mentioned works are applicable in the context of point sets. We will briefly list them in the following paragraph.

Gumhold, Wang, and MacLeod [GWM01] use a Riemannian graph on the point set to obtain connectivity information. Utilizing this, they evaluate a correlation ellipsoid at each point. Depending on the shape, the point under consideration is labeled as a surface-, crease-, border-, or corner-point. Pauly, Keiser, and Gross [PKG03] utilize a multi-scale approach to make feature detection with principal component analysis more resilient against noise. They compute the eigenvectors of the covariance matrix for up to 200 neighborhood sizes and consider the variations in the results. According to these, points are labeled as features. For feature reconstruction, a minimum spanning graph on the identified points is used. Lange and Polthier [LP05] consider an approach of Taubin [Tau95] that has originally been introduced for meshed geometries. In their paper, the authors translate this approach to point sets in the context of anisotropic smoothing. A curvature estimate is used to distinguish features from flat regions and thus only the latter are smoothed. Demarsin et al. [Dem+07] first segment the point set into clusters with equal normal behavior. A graph on these clusters indicates by its edges the presence of sharp features. Once more, a minimum spanning tree is used to close and reconstruct the features. Park, Lee, and Lee [PLL12] use the concept of tensor voting for feature detection. After the application of the voting tensor, an optimal scale is selected. These two steps are iterated until a user-given threshold is reached. Based on this, features are classified and completed.

Several methods are available that introduce feature-awareness to the MLS framework. The *Robust Implicit MLS* (RIMLS) of Öztireli, Guennebaud, and Gross [ÖGG09] uses non-linear kernel regression and an iterative scheme to determine the minimum of the moving least squares problem. Within these iterations, normal similarity is taken into account in order to reconstruct the surface with respect to the features. A method for feature-aware resampling of point sets based on MLS is presented by Huang et al., see [Hua+13]. The authors first resample the point set away from features following the work of Lipman et al., see [Lip+07]. Then, in a second step, features—as identified by their normal variation—are resampled. Given these examples, it becomes obvious that the MLS procedure can be practically used in the context of feature processing.

Finally, in Section 8, a feature detection based on the concept of tensor voting, comparable to

the approach of Park, Lee, and Lee [PLL12] is used to detect features before denoising the point set.

All these approaches work well in practice, but they have no mathematical guarantees. In fact, they are rather ad hoc constructions than theoretically based.

7.2 The Feature Detection Method

The MLS procedure has already been presented in Section 4.4 with several results. By the results of [SL16], an application of the MLS procedure yields a smooth manifold. However, when applied to a point set sample of a geometry with sharp features, the MLS result will still give a smooth version of this geometry, losing the features. Therefore, several publications aim at including the preservation of sharp features in the MLS technique. Fleishman, Cohen-Or, and Silva [FCS05] utilize the framework of robust statistical methods. They consider an initial robust estimator, combined with a forward search on the point set to locally classify piecewise smooth surface patches. Features are then derived as intersections of these smooth parts. Öztireli, Guennebaud, and Gross [ÖGG09] also use a statistical tool, namely *kernel regression*. They localize this technique robustly to obtain a signed scalar field from which the surface can be derived. See Section 1.5.2 for our version of feature-aware MLS.

Both these methods—[FCS05] and [ÖGG09]—work well in practice, but they depend on the statistical concepts added to the MLS procedure. Therefore, they depend on the actual parameters chosen and do not benefit from the mathematical guarantees of MLS as given by Theorem 13. In this section, we aim at a feature detection method that only needs the MLS procedure and its related quantities and is thus directly motivated by the main theoretical results from [Lev04; SL16].

We make the following central observation about Theorem 13. Its statement applies to some smooth manifold $\mathcal{M}$ with a suitable sampling P. Thus, given an arbitrary geometry that is piecewise comprised of smooth manifold patches but also includes sharp features, we know that Theorem 13 will still hold for all those areas that are locally smooth. However, from the theorem it remains unclear how the MLS procedure will perform close to sharp features, where the first derivative is discontinuous. The proof of Theorem 13, given in [SL16], is based in large parts on the work in [Lev98]. There, the author first established the approximation order in a comparably reduced setting, [Lev98, Theorem 5, p. 1523]. The proof is based on the observation that the coefficients of the Taylor expansion of the approximated function are bounded. This is true for smooth parts of the geometry, but is violated at feature points. Therefore, the MLS procedure will not necessarily satisfy the approximation order as the Taylor expansion is not well-defined in the first or second derivative. In the following, we will use this observation. Namely, we will search for points in the geometry that do not behave according to Theorem 13 and label these as features.

In the upcoming sections, we present four different approaches that each identify those areas of the geometry which do not behave according to the predicted approximation order. The approaches presented detect features with increasing sensitivity and can be used according to the needs of the algorithm in use. Note that everything that is marked by the following approaches is a feature in this sense, but there are features that will not be detected, e.g. at a sharp saddle point where the saddle is projected to itself under the MLS procedure. However, in our experiments, all major features of the geometries were detected, see Section 7.3.

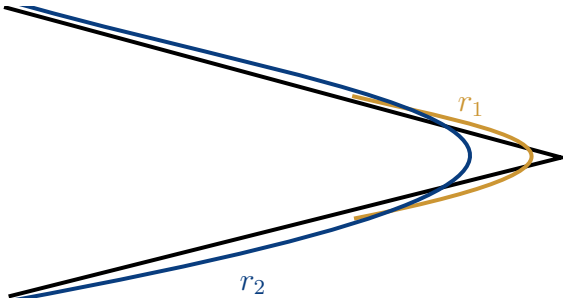


Figure 7.1: *MLS approximation with two different radii $r_1 < r_2$ at a corner. Note how the approximation with the larger radius r_2 is pulled away from the corner.*

7.2.1 Two Radii

When performing the MLS procedure, the non-negative weight function θ used in Equations (4.2) and (4.4) is using the radius r of its local support. On a small patch of the geometry resembling a smooth manifold, varying the radius will have negligible effects, as the local approximation will stay close to the smooth patch. However, at a feature point, enlarging the radius and thus taking more points away from the feature into account will draw the approximation further away from the feature, see Figure 7.1. Therefore, at a given point $p_i \in P$, we consider the value

$$\mu^{2rad}_{r_1,r_2}(p_i) = \left\| \mathbb{MLS}_{P,r_1}(p_i) - \mathbb{MLS}_{P,r_2}(p_i) \right\|, \tag{7.1}$$

such that we investigate the effect that a change of radius has on the approximation at point p_i. Note that the two used radii r_1 and r_2 cannot be chosen completely arbitrarily. There have to be enough points in the respective neighborhoods such that the linear system resulting from Equation (4.4) can be solved. When implementing the quantity, the two parameters could be chosen either globally or locally. We elaborate on our choice of r_1 and r_2 in Section 7.3.

7.2.2 Double Projection

The MLS procedure is a projection as established in [Lev04] and stated in Theorem 12. That is, when applying the operator $\mathbb{MLS}$ (4.7) to a point p twice, taking into account the point set P and a radius r, it holds that

$$\mathbb{MLS}_{P,r}(\mathbb{MLS}_{P,r}(p)) = \mathbb{MLS}_{P,r}(p). \tag{7.2}$$

We denote the set of projections of all sample points by

$$P' := \{\mathbb{MLS}_{P,r}(p_i) \mid p_i \in P\}.$$

Then, we will consider a second projection of the $p'_i := \mathbb{MLS}_{P,r}(p_i)$, not with respect to P, but with respect to P'. Note that because of property (7.2), if projecting a second time with respect to P, the points would not move at all. Thus, our second projection of $p'_i \in P'$ is performed with respect to P' and we denote it by

$$p''_i = \mathbb{MLS}_{P',r}(p'_i).$$

For all those points p_i sampled from smooth manifold patches of the geometry, p' will approximate the manifold well according to Theorem 13. Thus, the distance $\|p'_i - p''_i\|$ will be negligible. However, for points close to features of the geometry, the distance will be larger than the distance of points on smooth manifold patches and we denote it by

$$\mu^{double}_{r}(p_i) = \left\| \mathbb{MLS}_{P,r}(p_i) - \mathbb{MLS}_{P',r}(p'_i) \right\|. \tag{7.3}$$

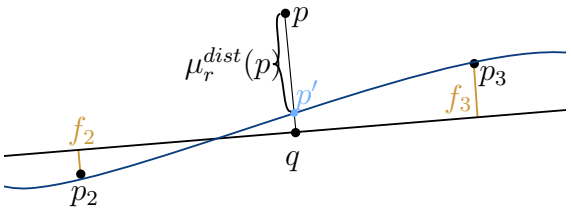


Figure 7.2: *Illustrating the value $\mu_r^{dist}(p)$ in the setting of Figure 4.4.*

7.2.3 Distance of the Original to the Projection

Consider a sample point $p_i \in P$—possibly affected by noise—that is taken from a smooth manifold patch of the geometry. When using the MLS procedure, the distance of p_i to its projection $\mathbb{MLS}(p_i)$ will be proportional to the level of noise σ and the approximation order $\mathcal{O}(h^{m+1})$, see Theorem 13. Consider a point sampled close to a feature of the geometry. When projecting this point, it will move not only according to the noise level and approximation order. It additionally has to move from the feature to a smooth manifold approximating the feature. Therefore, points close to features will have a larger distance to their projections than those points on smooth manifold patches. We consider the value

$$\mu_r^{dist}(p_i) = \|p_i - \mathbb{MLS}_{P,r}(p_i)\|\,, \tag{7.4}$$

see Figure 7.2 for an illustration.

7.2.4 Value of MLS functional

Finally, we consider the value of the MLS functional $I_p(a,s)$ as given in Equation (4.2) in order to distinguish features from non-feature points. The functional evaluates how well a given point p and its neighbors $p_i \in P$ can be approximated by a hyperplane. By definition, a small neighborhood of a smooth manifold resembles a flat Euclidean area. Therefore, the value of the function $I_p(a,s)$ will be small for those points p sampled from a smooth manifold patch of the geometry. However, a feature cannot be approximated well by a hyperplane. Therefore, points sampled close to features will obtain higher values in the MLS functional. We denote this last value by

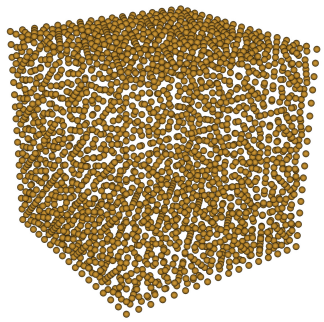
$$\begin{aligned}\mu_r^{Func}(p_i) = &\min \sum_{j=1}^{n} (\langle a, p_j\rangle - \langle a, p_i + sa\rangle)^2 \cdot \theta(\|p_j - (p_i + sa)\|) \\ &\text{s.t. } \|a\| = 1.\end{aligned} \tag{7.5}$$

7.2.5 Normalization

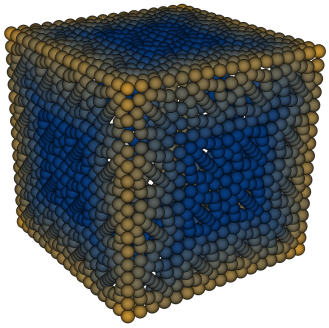
The four values introduced above—μ_{r_1,r_2}^{2rad}, μ_r^{double}, μ_r^{dist}, and μ_r^{Func}—have very different ranges. In order to compare them and use them in our experiments, we perform the following normalization

$$\bar{\mu}_\beth(p_i) = \frac{\mu_\beth(p_i) - \min_{j=1}^{n} \mu_\beth(p_j)}{\max_{j=1}^{n} \mu_\beth(p_j) - \min_{j=1}^{n} \mu_\beth(p_j)}, \tag{7.6}$$

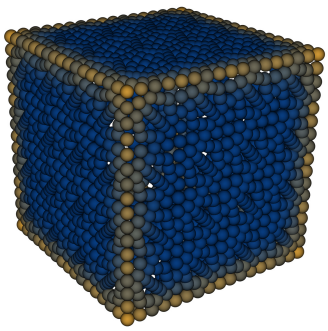
where $\beth \in \{2rad, double, dist, Func\}$. Note that the four different feature measures presented in the section are all based on intrinsic quantities of the MLS framework, linked to its central results as given in Theorems 12 and 13.



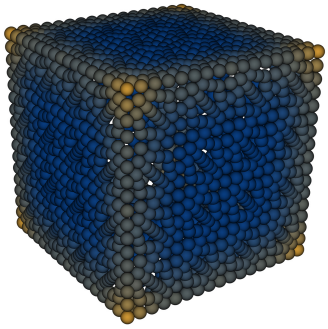
(a) *Original Cube with* 2306 *points.*



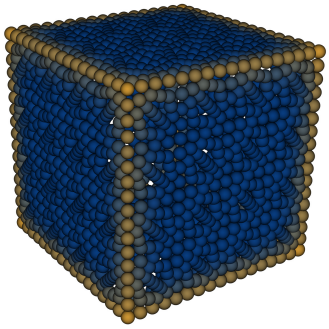
(b) *Principal curvature.*



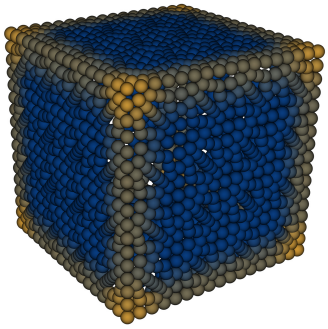
(c) *Two radii* $\bar{\mu}^{2rad}_{r_1,r_2}$, $r_1 = 0.3$, $r_2 = 0.1$.



(d) *Double projection* $\bar{\mu}^{double}_{r}$.



(e) *Projection distance* $\bar{\mu}^{dist}_{r}$.



(f) *MLS functional* $\bar{\mu}^{Func}_{r}$.

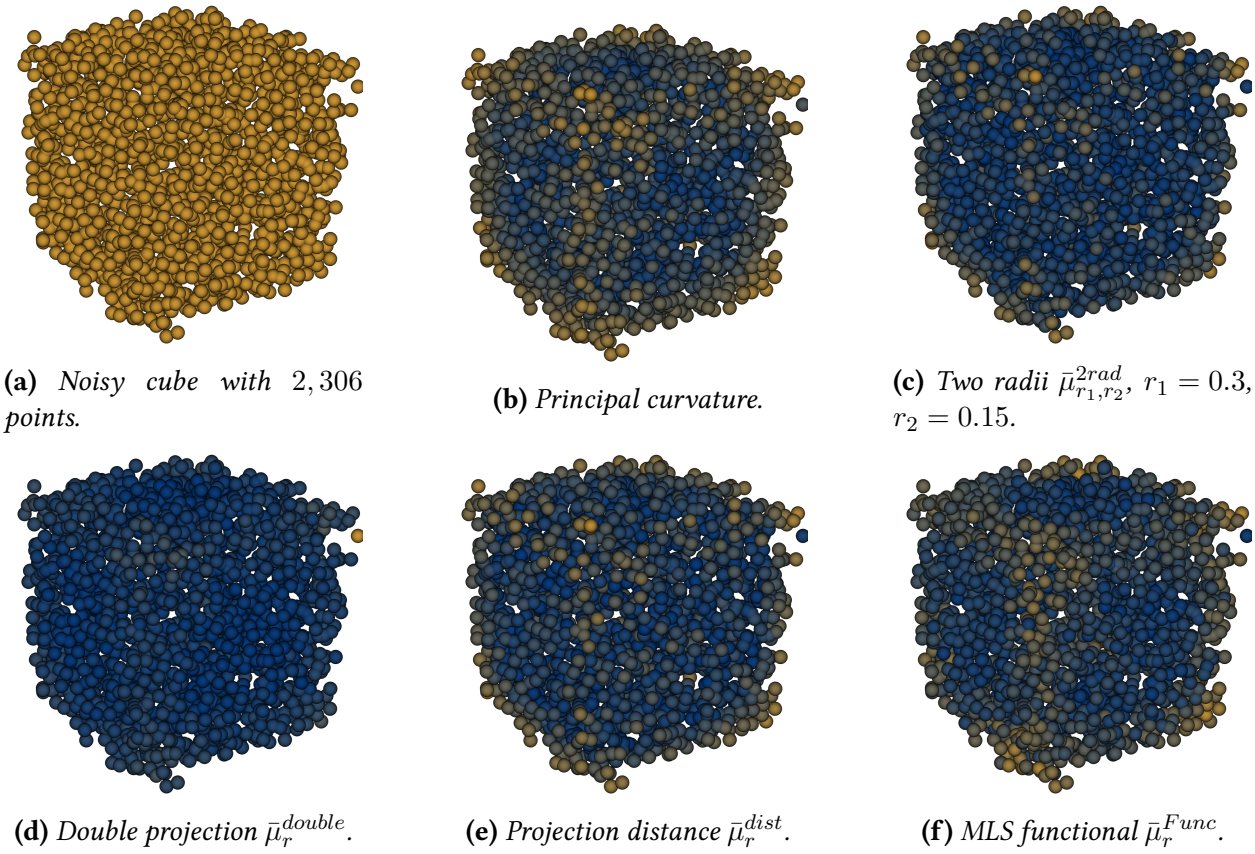
Figure 7.3: *Cube model colored according to the different schemes, linearly from blue (minimum) to yellow (maximum).*

7.3 Experimental Results

We evaluate our proposed approaches on several models. Both synthetic and real-world models as well as clean and noisy models are taken into account. Furthermore, we compare the result of our approaches to an evaluation of principal curvature on the models. For the principal curvature computation, we follow the procedure originally introduced by [Tau95] and translated to the setting of point sets by [LP05]. In all our experiments, we determine the radius r as a global parameter such that for each point p_i of the geometry, there are at least 30 points p_j in its neighborhood that satisfy $\theta(\|p_i - p_j\|) > 0.1$. We choose 30 points, as we are using a total polynomial degree of $m = 3$ and therefore the linear system following from Equation (4.4) has ten equations. In order to have enough points with a substantial contribution to the system, we use three times that minimally needed number. Thereby, we ensure that the system can always be solved uniquely. For the values $\bar{\mu}^{2rad}_{r_1,r_2}$, we use $r_1 = r$ as described above and $r_2 = r_1/3$, which is possible, as the margin to solve the linear system was chosen large enough.

7.3.1 Synthetic Models

Noiseless Our first testing model is a noise-free unit-cube, sampled with $2,306$ points, see Figure 7.3a. We pick this model as it symmetrically incorporates planar faces, edges, and corners. Thus, it provides a suitable first benchmark. The radius determined according to the procedure as explained above is $r_1 = 0.3$, with $r_2 = 0.1$. The original model, as well as the features detected by principal curvature computation and the four presented approaches is given in Figure 7.3. Note



(a) *Noisy cube with* $2,306$ *points.*

(b) *Principal curvature.*

(c) *Two radii* $\bar{\mu}_{r_1,r_2}^{2rad}$, $r_1 = 0.3$, $r_2 = 0.15$.

(d) *Double projection* $\bar{\mu}_r^{double}$.

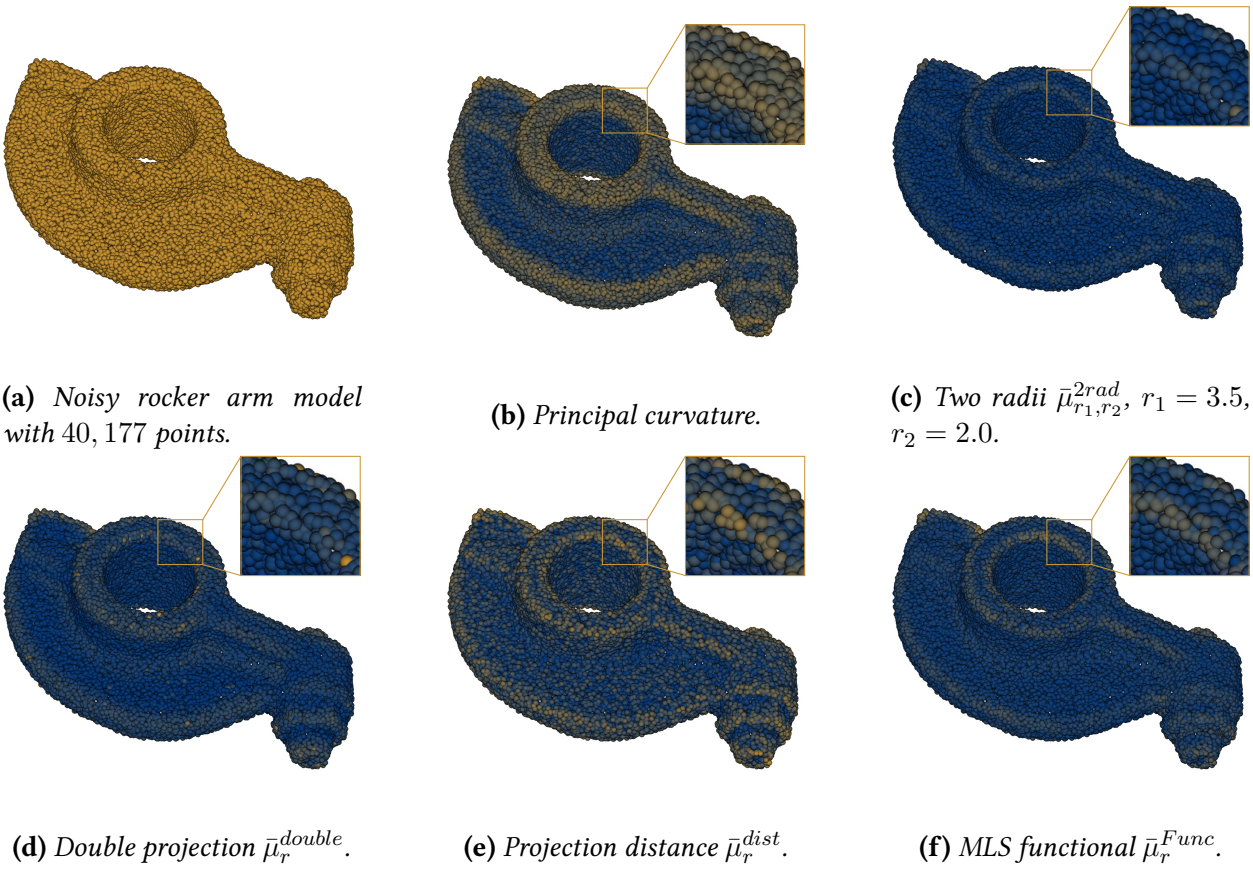
(e) *Projection distance* $\bar{\mu}_r^{dist}$.

(f) *MLS functional* $\bar{\mu}_r^{Func}$.

Figure 7.4: *Noisy cube model* ($\sigma = 0.1$) *colored according to the different schemes, linearly from blue (minimum) to yellow (maximum).*

how the proposed methods detect features of the cube with varying sensitivity. Both $\bar{\mu}_{r_1,r_2}^{2rad}$ and $\bar{\mu}_r^{dist}$ highlight small neighborhoods around both corners and edges of the cube. However, they do not distinguish significantly between corners and edges. The other two proposed methods $\bar{\mu}_r^{double}$ and $\bar{\mu}_r^{Func}$ highlight larger areas around the features, but corners and edges are visually easily separated. Compared to the principal curvature computation, all four methods distinguish better between the edges and the flat sides of the cube.

Noisy In order to test the robustness of our proposed methods, we apply artificial tangential and normal noise to the cube model of Figure 7.3. The applied noise has a standard deviation of $\sigma = 0.1$, with the original point set being a unit-cube. For the radii, we use slightly different values $r_1 = 0.3$ and $r_2 = 0.15$ compared to the clean case. This is necessary as the linear system resulting from Equation (4.4) cannot be solved for $r_2 = 0.1$. The noisy model, as well as the versions colored by features are given in Figure 7.4. Note how in the case of principal curvature and the quantity $\bar{\mu}_r^{double}$ mostly outliers attain the highest values, while in the colored models with quantities $\bar{\mu}_r^{dist}$ and $\bar{\mu}_r^{Func}$, the corner areas are highlighted well. Furthermore, while the principal curvature operator detects moderate values of curvature also on the flat sides of the cube, both $\bar{\mu}_{r_1,r_2}^{2rad}$ and $\bar{\mu}_r^{Func}$ identify comparably large areas on the sides as flat. While $\bar{\mu}_r^{double}$ completely fails in this presence of higher noise levels, the fourth approach $\bar{\mu}_r^{dist}$ is also less resilient to the high level of noise, but produces results that are visually comparable to those obtained by the principal curvature values.



(a) *Noisy rocker arm model with* $40,177$ *points.*

(b) *Principal curvature.*

(c) *Two radii* $\bar{\mu}_{r_1,r_2}^{2rad}$, $r_1 = 3.5$, $r_2 = 2.0$.

(d) *Double projection* $\bar{\mu}_r^{double}$.

(e) *Projection distance* $\bar{\mu}_r^{dist}$.

(f) *MLS functional* $\bar{\mu}_r^{Func}$.

Figure 7.5: *Noisy rocker arm model* ($\sigma = 0.5$) *colored according to the different schemes, linearly from blue (minimum) to yellow (maximum).*

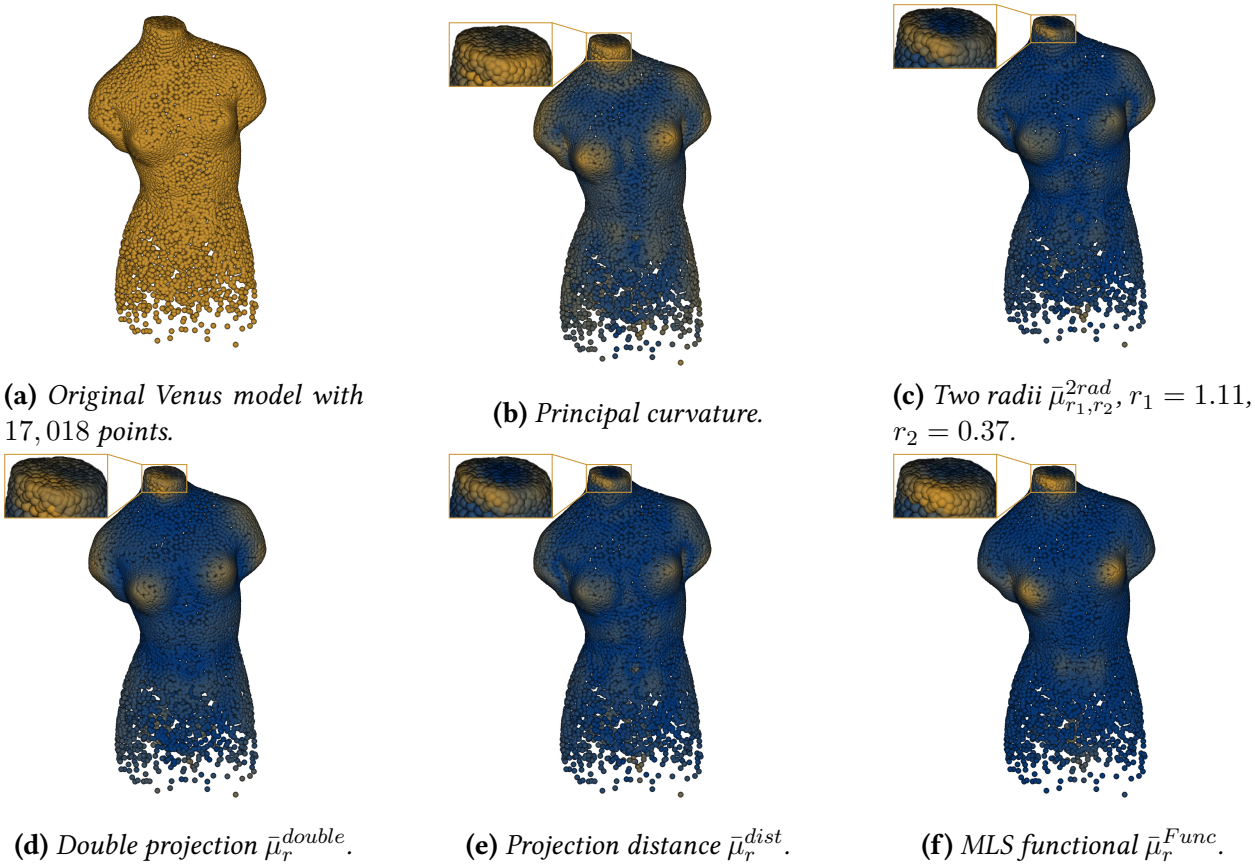
7.3.2 Real World Models

Noisy CAD Model In terms of real-world models, we first consider a noisy model of the rocker arm, see Figure 7.5. Note that while the principal curvature values detect the whole rim of the cylindrical hole as curved, our proposed quantities distinguish between the border of the rim and the actual flat rim surface. The same effect can be observed at the large curved part at the front of the model. None of the used quantities is affected by the presence of noise.

Scanned Data Finally, we compare the proposed feature detection mechanisms and principal curvature computation on raw scanning data of the Venus model. The density of the point set is getting smaller towards the bottom of the model. This effects the principal curvature weights to the extent that many curved points are detected simply because of the low density. The four proposed approaches are significantly less affected by the sparse sampling. Furthermore, in particular for values $\bar{\mu}_{r_1,r_2}^{2rad}$ and $\bar{\mu}_r^{dist}$, the flat area above the neck is well distinguishable from the curved rim of the neck, which is not the case for the other methods.

7.4 Conclusion and Future Work

In this section, we have presented four different approaches how to implement feature detection in the framework of MLS. In contrast to other algorithms, our techniques do not use additional statistical methods, but utilize the quantities generated within the MLS procedure itself. Our approaches are therefore based on the mathematical results and guarantees of the MLS technique. In



(a) *Original Venus model with* $17,018$ *points.*

(b) *Principal curvature.*

(c) *Two radii* $\bar{\mu}_{r_1,r_2}^{2rad}$, $r_1 = 1.11$, $r_2 = 0.37$.

(d) *Double projection* $\bar{\mu}_r^{double}$.

(e) *Projection distance* $\bar{\mu}_r^{dist}$.

(f) *MLS functional* $\bar{\mu}_r^{Func}$.

Figure 7.6: *Venus model colored according to the different schemes, linearly from blue (minimum) to yellow (maximum).*

several experiments, we showed that our techniques detect features with different sensitivity and are robust to noise. It remains as future work to investigate whether the presented approaches can be used to obtain a feature-aware version of the MLS algorithm, cf. Section 1.5.2. Furthermore, a thorough comparison of the presented quantities with state-of-the-art feature detection techniques has to be performed, which is also left as further research.

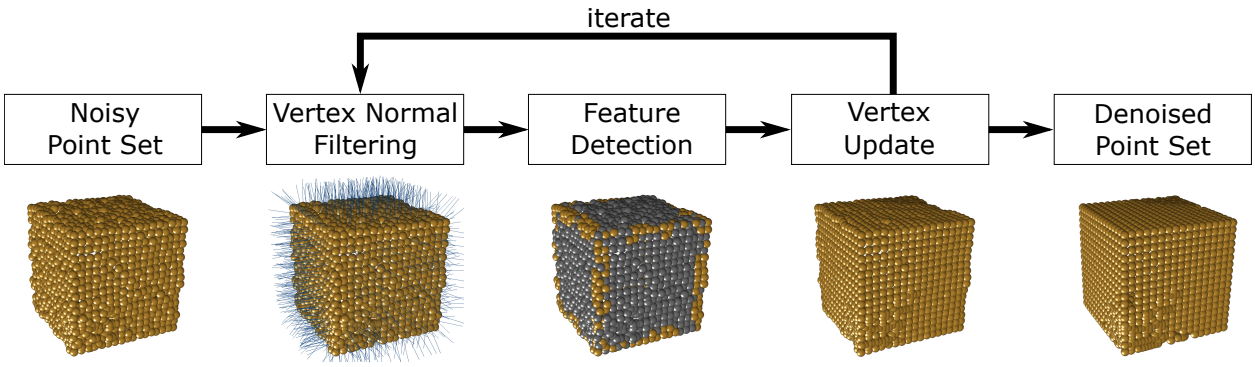


Figure 8.1: *The pipeline of the proposed algorithm. Our method consists of three different stages, which are applied iteratively.*

8 Constraint-Based Point Set Denoising using the Normal Voting Tensor and Restricted Quadratic Error Metrics

During the acquisition process of point sets, due to mechanical limitations and surrounding conditions, noise and outliers are inevitably added to the point set. These artifacts have to be removed in a post-processing step to obtain a cleaned point set, which can be used in further steps like surface reconstruction, computer aided design (CAD), or 3D printing. There exists a variety of denoising methods focused on removing outliers and noise from the input point set to create a high fidelity output. These methods do not only aim at removing the undesired components, but also try to preserve sharp features of the geometry. High frequency components like corners or edges should be preserved and not be smoothed out. This is a challenging task as both features and noise are high frequency components and thus ambiguous in their nature.

Most state-of-the-art denoising methods are designed to work on triangle meshes. Compared to this setup, working on point sets and preserving sharp features is more difficult as explicit connectivity information is not present. Also, we assume the input to be given without any normals. However, as point sets take up less storage space and as both surface reconstruction and point set registration are easier on a noise-free point set, we aim for an intrinsic smoothing method to work directly on the noisy point set input.

On a noisy point set, it is a challenging task to decouple noise components and sharp features, which is essential for a noise-free point set reconstruction. As shown in Figure 8.1, our algorithm consists of three different stages, which are iteratively applied until a satisfactory output has been computed. In the first stage, we extend the concept of face normal processing of Yadav et al. [YRP17] to the more general setup of vertex normal processing. In terms of noise sensitivity, vertex normals are more sensitive compared to face normals. Therefore, we modify the weighting scheme in the neighborhood selection of the *normal voting tensor* (NVT) to make the algorithm robust against different levels of noise. Noise and sharp features are decoupled using a spectral analysis of the vertex-based NVT and noise components are suppressed using *binary eigenvalue optimization* (BEO). In the second stage, we introduce an anisotropic covariance matrix using the filtered vertex normals to detect feature points (edges and corners) robustly on the noisy input point set. In the third and last stage, we update the vertex positions based on quadratic error metrics. A corresponding quadratic error metric is used based on different feature points. The proposed vertex update method helps the algorithm to preserve sharp features with minimum shrinkage during the denoising process.

The main contributions of this section are:

- Translation of the algorithm from [YRP17] to the context of point sets.
- Introduction of quadratic error metrics for the vertex update stage in the smoothing pipeline.

The results of this section have been presented at the SMI'18 conference and published in the corresponding proceedings, see list of publications prior to the thesis, page 5. Additional to the results published, this section includes several explanations concerning the motivation behind choices made in the publication—like choosing the eigenvalue optimization to work with binary values. Furthermore, the example in Figure 8.3 was added with corresponding explanations as well as a pseudo-code version of the presented algorithm.

8.1 Related Work

8.1.1 Point-based Methods

In general, point sets appear as natural output of 3D scanning devices. The increase in computational costs while processing polygonal meshes with growing size is partly responsible for the recognition of points sets as primitives for surface representation, cf. [AK04]. One major drawback in this approach is the absence of connectivity information, which sets the task to declare surface normals. Here, [AK04] especially proposes a definition utilizing surfels, which are points equipped with normals. Usually, point sets do not carry normals, so we have to rely on methods which determine these robustly and with high quality. The authors Mitra and Nguyen [MNG03] suggest a calculation of point set normals and an analysis under consideration of density, neighborhood sizes, and the presence of noise.

We are interested in point set denoising coupled with feature preservation. There are several works approaching these two properties directly. A first one was published by Fleishman et al. [FDC03]—serving as a representative despite the fact that it deals with meshes instead of point sets. As it does not use any specific mesh information, it can be transferred to the point set setting easily. They use a bilateral filtering of points in normal direction in local neighborhoods. Another one is the anisotropic smoothing of point sets of [LP05] where the authors use an anisotropic geometric curvature flow. Besides the high dependency on suitable neighborhoods, which the authors cannot compute directly, the proposed algorithm does not detect features explicitly, but incorporates feature detection into an anisotropic Laplacian. The more recent work [SSW15] is based on the idea of sparsity methods and includes L_0 minimization. Originating from image denoising, they set up an energy—consisting of the 3D signal to be optimized—coupled with an L_0 optimization applied to a differential operator on the signal.

Processing of normals, point positions, and an edge-aware upsampling offers the opportunity for an iterative application. In this setting, we are going to compare our algorithm with that of [MC17], called *moving robust principal component analysis* (MRPCA). The idea is—like the previous—based on sparsity methods, which takes sparsity-algorithms and adapt them to geometry processing problems. They perceive the point set as a collection of overlapping two-dimensional subspaces and do not rely—in contrast to other procedures—on oriented normals as input. The method is robust against outliers and capable of denoising the point set while handling sharp features.

Recently, Zheng et al. [Zhe+17] proposed an extension of edge-aware image processing and mesh denoising to point sets. In their four-staged approach, feature candidates are detected by employing a feature structure by the ℓ_1-medial skeleton, calculating and equipping these with multiple normals, and selecting guiding normals by using k nearest neighbor patches with its

normals being most consistent. In these terms, the algorithm is even capable of handling high intensive noise while preserving important geometric features.

8.1.2 Surface Reconstruction with Feature Preservation

One of the processes most affected by noise and outliers in a point set is that of surface reconstruction. A thorough introduction is given in the survey of Berger et al., see [Ber+17]. All following techniques aim at preserving features while simultaneously performing denoising in the surface reconstruction process. In the context of local smoothness priors, the moving least squares (MLS) approach has a major impact. Developed in large parts by Levin [Lev04], MLS underwent a lot of modifications. See Section 4.4.1 for an introduction to the MLS framework. Guennebaud et al. [GG07] modified the MLS idea by replacing the concept of finding well-defined tangent planes by fitting spheres as higher order approximations to the surface. This change makes the method more robust—especially in sparsely sampled regions, where a well defined tangent plane might not exist. Their method is denoted as *algebraic point set surfaces* (APSS) and will serve as comparison to our algorithm. The method of Öztireli et al. [ÖGG09] aims at overcoming the sensitivity of MLS to outliers and the effect of smoothing out small or sharp features. They combine MLS with local kernel regression to create a new implicit description of the surface, making it robust to noise, outliers, and even sparse sampling. Their method of *robust implicit moving least squares* (RIMLS) will be the third algorithm we compare to. More recently, Chen et al. [Che+13] set their focus on a new MLS formalism using higher-order approximations—like APSS—incorporating discrete non-oriented gradient fields, yielding a continuous implicit representation.

Turning to hierarchical partitioning, Ohtake et al. [Oht+03] propose *multi-level partitioning of unity implicits* (MPU). Their technique consists of an octree-based top-down structure, where points in a cell and nearby are approximated by either a bivariate quadratic polynomial or an algebraic trivariate quadric. An adjustment parameter for the level of smoothness guarantees the handling of noise with respect to an error residual tolerance.

Considering piecewise smooth priors and partition based methods, Fleishman et al. [FCS05] concentrate with their *robust moving least squares* (RMLS) on the handling of sharp features. They use the robust statistics tool of forward-search paradigm to choose small sets of points excluding outliers, continue through the point set, and evaluate observations monitored by statistical estimates. Wang et al. [Wan+13] robustly compute a feature preserving normal field by mean-shift clustering and a *least median of squares* (LMS) regression scheme, providing local partitions, to which edge-preserving smoothing is applied by fitting multiple quadrics. Due to the locality, feature fragmentation at sharp edges may occur.

Taking sparsity and neighboring normals into account, Avron et al. [Avr+10] use global ℓ_1 optimization on these normals, observing that differences between them should be sparse, yet large values should reflect sharp features. Similar to the approach in RIMLS, [Hua+13] suggests the *edge-aware resampling* (EAR) of the point set. This is a feature-sensitive method under the guidance of the *locally optimal projection* (LOP) of Lipman et al., see [Lip+07], in a two-staged approach, starting their robust smoothing and resampling process in regions with similar normal distribution, while approaching the edges in terms of both smoothing and resampling in a second step.

8.2 The Proposed Method

Let us consider a non-uniformly sampled and noisy input point set $P = \{p_i \in \mathbb{R}^3 \mid i = 1, \dots, n\}$ sampling a surface $\mathcal{M} \subset \mathbb{R}^3$ with $n \in \mathbb{N}$ denoting the number of points. We assume these

data points to be acquired e.g. by a 3D laser scanner and not to be equipped with vertex normals. Thus, a first normal field on the vertices is computed following [Hop+92], which results in consistently[2] oriented normals. Despite the fact that there are more recent works dealing with consistent normal fields on point sets, we decided to use [Hop+92], as the implementation is simple and it works well with all the models we used for our experiments. This is mostly due to the fact, that we process and smooth the normals further, so a consistent initial normal field is sufficient for our purposes. We denote the normal at vertex p_i by n_i and the normal field by $N_P = \{n_i \in \mathbb{R}^3 \mid p_i \in P\}$. For a given vertex p_i, we denote by $\mathcal{N}_\varepsilon(p_i)$ those points from P that have distance less or equal ε to p_i, see (1.4), where ε is a global parameter. We favor a geometric neighborhood over a combinatorial k nearest neighborhood (1.1) as [YRP17] found it to be more robust. In the following, we will indicate by a tilde $\sim$ those elements updated in one iteration—e.g. updated normals $\tilde{n}_i$—which will then serve as input to the next one.

8.2.1 Vertex Normal Filtering

This is the first of three iteratively applied steps of the proposed method. Here, noisy vertex normals are filtered and denoised using a vertex-based *normal voting tensor* (NVT) and *binary eigenvalues optimization* (BEO) similar to [YRP17]. We describe both in the following.

Vertex-based Normal Voting Tensor (NVT) Covariance matrices compute the variance of an entity in a well-defined domain. For example, consider a vertex p_i and its nearest neighbors $p_j \in \mathcal{N}_\varepsilon(p_i)$. The covariance matrix on the edges $(p_j - p_i)$ of the nearest neighbor graph computes the variance of the vertex p_i in $\mathbb{R}^3$. Similarly, the covariance matrix of the vertex normals n_j corresponding to the vertices p_j, in a well defined neighborhood computes the anisotropic nature of a point set in that region. To analyze the anisotropic nature of a shape—in this case a point set surface—we define an object related to the covariance matrix, namely the vertex-based NVT, which is computed using the neighboring vertex normals:

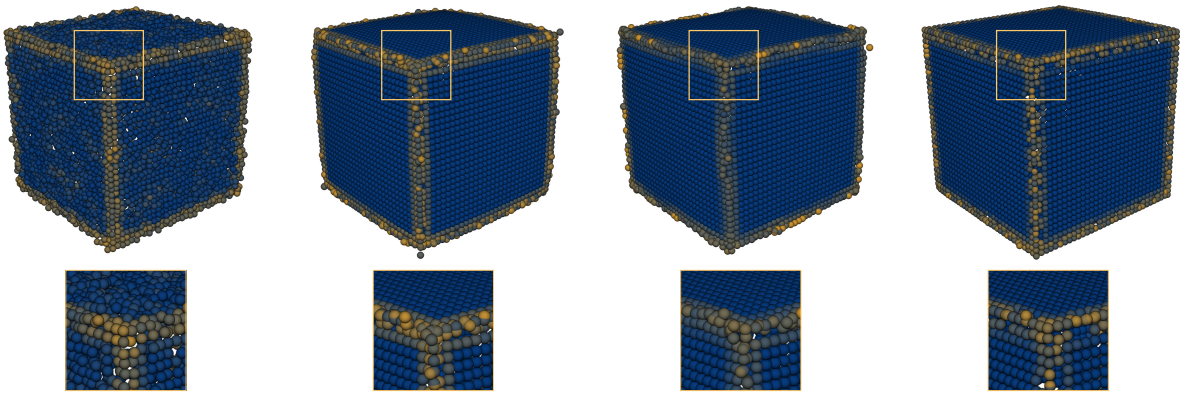
$$T_i = \frac{1}{|\mathcal{N}_{\varepsilon,\rho}(p_i)|} \sum_{p_j \in \mathcal{N}_{\varepsilon,\rho}(p_i)} n_j \otimes n_j, \tag{8.1}$$

where the tensor $n_j \otimes n_j$ can be represented by the outer product $n_j n_j^T$. This NVT has been used as vertex-based NVT in [YRP17] for mesh denoising and was introduced as element-based NVT in [KCL09]. In the latter, each summand $n_j \otimes n_j$ was weighted by a term depending on the area of the planar element considered. The authors of [YRP17] introduce a different weighting scheme that also depends on the face areas, but also on the similarity of the normals. As area weights are not available in our mesh-free context, we weight all summands with uniform weights equal to 1 and thus normalize by the factor $1/|\mathcal{N}_{\varepsilon,\rho}(p_i)|$ to become resolution independent. However, this is not strictly necessary as we will only work with the spectral components of T_i in the following, which are not affected by the scalar factor. More elaborate weights could be investigated which better reflect the density of the point set, see Section 6. However, in all our experiments the utilized weights work reasonably well.

In Equation (8.1), the set $\mathcal{N}_{\varepsilon,\rho}(p_i)$ is a binary cutoff neighborhood given by

$$\mathcal{N}_{\varepsilon,\rho}(p_i) = \{p_j \in P \backslash \{p_i\} \mid w_{ij}^{bin} = 1\} \cap \mathcal{N}_\varepsilon(p_i), \tag{8.2}$$

[2] Consistency in this case refers to the following property of the algorithm: If the underlying surface $\mathcal{M}$ is orientable, given a good-enough sampling, the normals will all be inward- or outward-normals after the application of algorithm [Hop+92].



(a) $\sigma_n = 0.25\ell_a$ **(b)** w_{ij} *as in [YRP17]* **(c)** *Bilateral [YRP18]* **(d)** *As in Eq. (8.3)*

Figure 8.2: *A comparison between different weighting functions for neighborhood selection. (a) The cube model is corrupted with a Gaussian noise ($\sigma_n = 0.25\ell_a$) in random directions, where ℓ_a is the average distance between vertices of the point set. (b) The output obtained by using the weighting function mentioned in Yadav et al. [YRP17]. (c) The output obtained by using the bilateral weighting function (smooth functions) mentioned in Yadav et al. [YRP18]. (d) The output obtained by using the proposed weighting function of Equation (8.3). Vertices of the cube are colored from blue (low) to yellow (high) according to the variation of vertex normals. While the method uses ε nearest neighbors (1.4), the cube model is sampled nicely, so the features shown were computed with k nearest neighbors (1.1), $k = 16$.*

with $\mathcal{N}_\varepsilon(p_i)$ a metric neighborhood as in Equation (1.4) and binary weights

$$w_{ij}^{bin} = \begin{cases} 0 & \text{if } \langle n_i, n_j \rangle < \rho \\ 1 & \text{if } \langle n_i, n_j \rangle \geq \rho \end{cases}. \tag{8.3}$$

Here, $\rho \in \mathbb{R}_{>0}$ is given by the user and denotes a local binary neighborhood threshold, which is used to select vertices $p_j \in \mathcal{N}_\varepsilon(p_i)$ with similar normals to n_i. The neighborhood is not the exact weighted neighborhood used in [YRP17] because vertex normals are more sensitive to noise than face normals, for example at sharp features. Therefore, a harder cut-off is necessary to maintain geometrical features while smoothing the point set. Figure 8.2 shows a comparison between the proposed weighting scheme (8.3), the bilateral weighting from [YRP18], and the weighting function used in Yadav et al. [YRP17]. As it can be seen, the harder cut-off neighborhood is more effective in terms of feature preservation than Yadav et al. [YRP17] or [YRP18]. The sharp cut-off can be formulated in terms of Equation (1.13) by choosing parameters $a = \rho$ and $b = \infty$ for the sigmoid. See Section 1.5.1 for a detailed discussion of the used parameters.

By construction, T_i is symmetric and positive semidefinite and can be represented in terms of its spectral components:

$$T_i = \sum_{\ell=1}^{3} \lambda_{i,\ell}\, x_{i,\ell} \otimes x_{i,\ell}, \tag{8.4}$$

where $\lambda_{i,\ell}$ and $x_{i,\ell}$ are the corresponding eigenvalues and eigenvectors. Let us consider the eigenvalues to be sorted in decreasing order $\lambda_{i,1} \geq \lambda_{i,2} \geq \lambda_{i,3} \geq 0$. Thus, we can rewrite T_i as:

$$\begin{aligned} T_i =& (\lambda_{i,1} - \lambda_{i,2}) x_{i,1} \otimes x_{i,1} + (\lambda_{i,2} - \lambda_{i,3})(x_{i,1} \otimes x_{i,1} + x_{i,2} \otimes x_{i,2}) \\ &+ \lambda_{i,3}(x_{i,1} \otimes x_{i,1} + x_{i,2} \otimes x_{i,2} + x_{i,3} \otimes x_{i,3}). \end{aligned} \tag{8.5}$$

Here, the first term of the right hand side is known as the stick tensor and has only one dominant eigenvalue in the normal direction. The second term is spanned by the two dominant eigenvectors, such that the normal direction is defined in the direction of the least dominant eigenvector. This term is known as the plate tensor. The third term is spanned by all eigenvectors and does not have a well defined normal direction, cf. [MTL00]. From the above description, it is clear that the vertex-based NVT captures the anisotropic nature of a point set and feature points can be easily detected using the eigenvalues of T_i. That is, if there is only one dominant eigenvalue then it is a planar point, if two eigenvalues are dominant then it is an edge, and if all eigenvalues are equally dominant then it is a corner.

Binary Eigenvalue Optimization (BEO) In our method, the vertex-based NVT is applied as a denoising operator on a noisy point set. Furthermore, as we have discussed in the last section, the vertex-based NVT is capable of detecting features on point sets as shown in Figure 8.1 (third column). However, on a noisy point set, the behavior of the spectral components of the vertex-based NVT will change.

Let us assume that a point set is corrupted by random noise with standard deviation σ. Due to the presence of noise, the eigenvalues of the vertex-based NVT will change. For example, on a planar area, one eigenvalue will remain dominant, but the other two eigenvalues will be non-zero and proportional to σ. Similarly, on an edge of the sampled geometry, the least dominant eigenvalue will be proportional to the applied noise, i.e. $\lambda_{i,3} \propto \sigma$. On a corner of the sampled geometry we expect $\lambda_{i,1}, \lambda_{i,2}, \lambda_{i,3} \gg \sigma$. To remove these noise effects from the vertex-based NVT, the eigenvalues of T_i should be modified accordingly. That is, on a planar area and on an edge the least dominant eigenvalues should be zero and at a corner all eigenvalues should be equally dominant. In order to achieve this, we turn to binary optimization.

The concept of binary eigenvalue optimization is applied to the eigenvalues of T_i, where each eigenvalue will be assigned a binary value $\tilde{\lambda}_{i,\ell} \in \{0, 1\}$ to remove noise components effectively. Similar to [YRP17], a threshold value $\tau \in \mathbb{R}_{>0}$ is used for the BEO. The term τ is a global parameter given by the user and should be chosen according to the noise intensity, i.e. $\tau \propto \sigma$, and to be smaller than the dominant eigenvalues of all T_i respectively. We will denote by $\tilde{\lambda}_{i,\ell}$ the modified eigenvalues of the vertex-based NVT after BEO. The modification is based on feature classification:

- At corners of the sampled geometry, when considering the point set, the smallest eigenvalue should still be bigger than the threshold value, i.e. $\lambda_{i,3} \geq \tau$. Hence, we set:

$$\tilde{\lambda}_{i,\ell} = 1, \quad \ell \in \{1, 2, 3\} \quad \text{if} \quad \lambda_{i,3} \geq \tau.$$

- At edges of the sampled geometry, in the noisy point set, the least dominant eigenvalue should be smaller than the threshold value, i.e. $\lambda_{i,3} < \tau$ and $\lambda_{i,2} \geq \tau$. Hence, we set:

$$\tilde{\lambda}_{i,1} = \tilde{\lambda}_{i,2} = 1, \quad \tilde{\lambda}_{i,3} = 0 \quad \text{if} \quad \lambda_{i,2} \geq \tau, \quad \lambda_{i,3} < \tau.$$

- In the last case, we check for planar areas of the geometry. Having $\lambda_{i,2} < \tau$ and $\lambda_{i,3} < \tau$ shows that the only dominant eigenvalue is $\lambda_{i,1}$. Hence, we set:

$$\tilde{\lambda}_{i,1} = 1, \quad \tilde{\lambda}_{i,2} = \tilde{\lambda}_{i,3} = 0 \quad \text{if} \quad \lambda_{i,1} \geq \tau, \quad \lambda_{i,3},\lambda_{i,2} < \tau.$$

The BEO procedure presented here will remove the noise components from the eigenvalues of the vertex-based NVT.

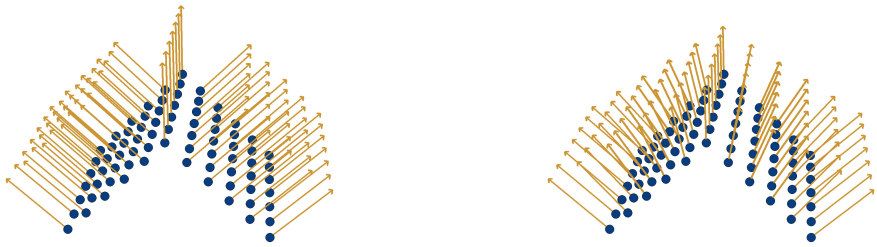


Figure 8.3: *Example for a geometry, where binary values of the eigenvalues $\lambda_{i,\ell}$ are necessary to retain the edge. See the right image, where the normal field is smoothed away despite the perfect input shown in the left image.*

Note that by changing the eigenvalues of T_i, we also change its trace significantly. One could consider to not set the eigenvalues to binary values, but to distribute the non-dominant eigenvalue(s) to the dominant one(s), thus keeping the trace. However, this will not provide the desired result as the following example shows: Consider a geometry with two planar face sharing a straight edge. Assume the dihedral angle between the two faces to be less than $90°$, see Figure 8.3. Then, the normal does not change along the edge. Therefore, the eigenvector pointing along the edge will be assigned eigenvalue 0. The other two eigenvectors will be orthogonal to the edge and will be derived from the normals of the two faces, depending on where exactly T_i is computed and how many normals of which face are taken into account.

When we now apply this NVT to all normals of the geometry, they all will change. Instead of the perfect normals from the input geometry, we get a smoother normal field, in particular around the edge, see Figure 8.3. When using binary eigenvalues instead, we obtain an NVT matrix which—when represented in the eigenbasis—is an identity matrix except for a single 1 missing, corresponding to the direction of the edge. However, all normals are orthogonal to this direction and thus remain unchanged. In an imperfect situation—i.e. on a geometry with noise—only the noise component parallel to the feature edge will be deleted, but that part of the normal information which determines affiliation to a planar area is not changed. In both the planar case and the corner case, the amplitude of the eigenvalues is irrelevant. Thus, given the example above, we are willing to change the trace of the NVT in order to retain edges.

Vertex Normal Denoising To remove noise components from the vertex normals, we project the noisy vertex normals towards smooth normals by multiplication of the vertex-based NVT to the corresponding vertex normal. By the preceding BEO, this multiplication procedure will suppress noise in weak eigendirections and will strengthen vertex normals in strong eigendirections.

Before multiplication, we have to recompute the modified vertex-based NVT by using the same eigenvectors with the eigenvalues optimized in the BEO:

$$\tilde{T}_i = \sum_{\ell=1}^{3} \tilde{\lambda}_{i,\ell}\, x_{i,\ell} \otimes x_{i,\ell}. \tag{8.6}$$

To remove noise, we multiply the corresponding vertex normal with the modified tensor $\tilde{T}_i$. The multiplication will lead to noise removal while retaining sharp features:

$$\tilde{n}_i = Dn_i + \tilde{T}_i n_i = dn_i + \sum_{\ell=1}^{3} \tilde{\lambda}_{i,\ell} \langle x_{i,\ell}, n_i \rangle x_{i,\ell}, \tag{8.7}$$

where $D \in \mathbb{R}_{>0}$ denotes a damping factor to control the denoising speed of the vertex normals. We use $D = 3$ for all experiments. Finally, the updated normal $\tilde{n}_i$ is normalized.

8.2.2 Feature Detection

This is the second of three iteratively applied steps of the proposed method. Here, we classify the point set into three different categories: corners, edges, and planar points. This will be done using the spectral analysis of a weighted covariance matrix. The idea to identify points and their features is substantiated in the follow-up treatment of point position updates in the upcoming third subsection where we use the notion of a quadratic error metric applied differently to the occurring feature-assigned points. A weighted covariance matrix is defined using filtered vertex normals, which makes the proposed algorithm more robust against feature points misclassification, which can lead to feature blurring artifacts.

To detect feature points on a point set with filtered vertex normals, we consider the weighted covariance matrix, [HBC11]:

$$C_i = \frac{1}{\sum_{p_j \in \mathcal{N}_\varepsilon(p_i)} \tilde{w}_{ij}} \sum_{p_j \in \mathcal{N}_\varepsilon(p_i)} \tilde{w}_{ij} (p_j - \bar{p}) \otimes (p_j - \bar{p}), \tag{8.8}$$

where the weights $\tilde{w}_{ij}$ are similar to Equation (8.3), but are now utilizing the filtered vertex normals of Section 8.2.1. The terms $\bar{p}$ and $\tilde{w}_{ij}$ are defined as:

$$\bar{p} = \frac{1}{\sum_{j \in \mathcal{N}_\varepsilon(p_i)} \tilde{w}_{ij}} \sum_{j \in \mathcal{N}_\varepsilon(p_i)} \tilde{w}_{ij} p_j, \quad \tilde{w}_{ij} = \begin{cases} 1 & \text{if } \measuredangle(\tilde{n}_i, \tilde{n}_j) \leq \rho \\ 0 & \text{if } \measuredangle(\tilde{n}_i, \tilde{n}_j) > \rho \end{cases}, \tag{8.9}$$

where the aforementioned global parameter ρ is used. Similar to the vertex-based NVT T_i, the weighted covariance matrix C_i is also a symmetric and positive semidefinite matrix and can be represented in terms of its spectral components:

$$C_i = \sum_{\ell=1}^{3} \mu_{i,\ell}\, y_{i,\ell} \otimes y_{i,\ell}, \tag{8.10}$$

where $\mu_{i,\ell}$ and $y_{i,\ell}$ are the corresponding eigenvalues and eigenvectors. Let us again consider the eigenvalues to be sorted in decreasing order $\mu_{i,1} \geq \mu_{i,2} \geq \mu_{i,3} \geq 0$. In the proposed method, they are used to classify the points as follows, utilizing the same threshold parameter τ as in Section 8.2.1:

- On a planar area, there will be two dominant eigenvalues and their corresponding eigenvectors should be spanning the tangent plane. The least dominant eigenvalue will be smaller than the feature threshold τ. Therefore, we classify planar points as

$$P_f = \{p_i \in P \mid \mu_{i,1}, \mu_{i,2} \geq \tau,\ \mu_{i,3} < \tau\}.$$

- On an edge, there will be one dominant eigenvalue and the corresponding eigenvector aligns with the edge direction. Therefore, we classify edge points as

$$P_e = \{p_i \in P \mid \mu_{i,1} \geq \tau,\ \mu_{i,2}, \mu_{i,3} < \tau\}.$$

- Finally, on a corner, either all eigenvalues are dominant or none of them is significant. Therefore, corner points are set to

$$P_c = \{p_i \in P \mid (\mu_{i,1}, \mu_{i,2}, \mu_{i,3} \geq \tau) \vee (\mu_{i,1}, \mu_{i,2}, \mu_{i,3} < \tau)\}.$$

Points at a corner, an edge, and planar points are represented in the following by $P_c = \{p_\cdot^c\}$, $P_e = \{p_\cdot^e\}$, and $P_f = \{p_\cdot^f\}$ respectively, such that we obtain the following disjoint union

$$P = P_c \dot{\cup} P_e \dot{\cup} P_f.$$

8.2.3 Constraint-based Vertex Position Update

In this final of three iteratively applied steps of the proposed method, we update the vertex positions. This is done utilizing distance-based constraints, where the resulting updated point set remains within a prescribed distance to the input noisy point set. To compute the optimal position of a vertex w.r.t. the smoothed vertex normal, restricted quadratic error metrics are used in this algorithm, inspired by the work of [GH97]. The restriction to the quadratic error metric is introduced based on the different feature points and the vertex position is updated utilizing distance-based constraints.

We allow the user to provide a parameter $\nu \in \mathbb{R}_{>0}$ bounding the maximum variance v_i between an initial noisy point and its corresponding iteratively updated point $\tilde{p}_i$.

Vertex Update at Corners Let us consider a point $p_i^c \in P_c$, labeled as corner point in Section 8.2.2. We will find its updated position $\tilde{p}_i^c$ by minimizing the following energy function:

$$\min_{\tilde{p}_i^c \in \mathbb{R}^3} \sum_{p_j \in \mathcal{N}_\varepsilon(p_i)} \|\tilde{n}_j \cdot (\tilde{p}_i^c - p_j)\|^2. \tag{8.11}$$

Each of the neighboring vertices p_j is equipped with a corresponding filtered normal direction $\tilde{n}_j$. Thus, we can associate a plane based at each neighboring vertex given by the normal direction. In an ideal case, these planes would all meet in a point—the exact position of the vertex p_i. But as noise is present and the planes will in general not intersect in a point, we define the error of the vertex $\tilde{p}_i^c$ as the sum of squared distances to these planes.

Note that we do not weight the neighbors in Equation (8.11), but take all $p_j \in \mathcal{N}_\varepsilon(p_i)$ into account equally. That is because we rely on the highly unstable intersection of multiple planes in $\mathbb{R}^3$. The more we take into account, the more likely it is for them to even out noise effects and give a faithful reconstruction of the corner.

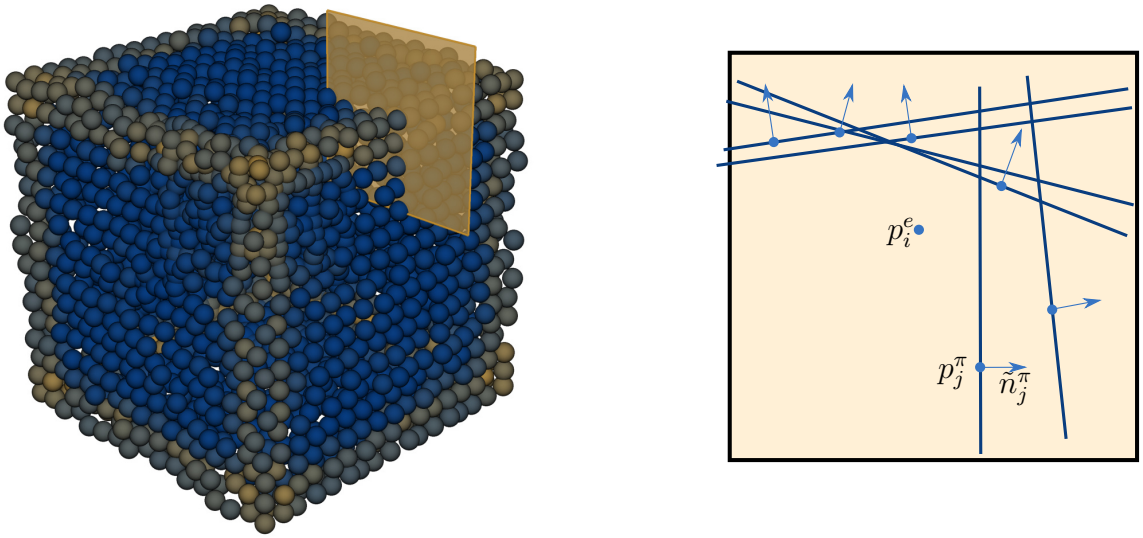
Minimizing Equation (8.11) boils down to solving a linear system and the new position can be computed directly, which is given by the following equation:

$$p_i'^c = \left(\sum_{p_j \in \mathcal{N}_\varepsilon(p_i)} \tilde{n}_j \otimes \tilde{n}_j \right)^{-1} \sum_{p_j \in \mathcal{N}_\varepsilon(p_i)} \left(\tilde{n}_j \otimes \tilde{n}_j p_j \right),$$

where $p_i'^c$ is a temporary vertex position. Before updating the position of p_i^c to $p_i'^c$, we compute the variation v_i between $p_i'^c$ and the corresponding original vertex from the noisy point set. If v_i is within the user-prescribed limit ν we move this corner point to p_i^c, otherwise we don't move this point:

$$\tilde{p}_i^c = \begin{cases} p_i'^c & \text{if } v_i \leq \nu \\ p_i^c & \text{if } v_i > \nu \end{cases}, \tag{8.12}$$

where ν is the aforementioned user input which limits the variation between the original noisy and updated corner points. By the above equation, corner points are moved at most ν during their position update in the direction of a minimum distance to all neighboring planes spanned by the respective normals.



(a) *The plane H_i, which is defined by y_1 for edge vertex p_i^e.*

(b) *Neighbor vertices of p_i^e and corresponding vertex normals are projected onto the plane.*

Figure 8.4: *A visual representation of the vertex update scheme at edges.*

Vertex Update at Edges Let us consider a point $p_i^e \in P_e$, labeled as edge point in Section 8.2.2. Here, the weighted covariance matrix C_i has only one dominant eigenvalue $\mu_{i,1}$ and the corresponding eigenvector $y_{i,1}$ aligns with the edge direction. We define a plane

$$H_i = \{p \in \mathbb{R}^3 \mid \langle y_{i,1}, p\rangle = \langle y_{i,1}, p_i^e\rangle\}.$$

As shown in Figure 8.4, we project all neighborhood vertices $p_j \in \mathcal{N}_\varepsilon(p_i)$ and their respective vertex normals $\tilde{n}_j$ to H_i, denoting the corresponding projections by p_j^π and $\tilde{n}_j^\pi$ (assuming that $\|y_{i,1}\| = 1$):

$$\begin{aligned} p_j^\pi &= p_j - \langle (p_j - p_i^e), y_{i,1}\rangle y_{i,1}, \\ \tilde{n}_j^\pi &= \tilde{n}_j - \langle \tilde{n}_j, y_{i,1}\rangle y_{i,1}. \end{aligned}$$

Now, we define a quadratic energy function similar to Equation (8.11):

$$\min_{\tilde{p}_i^e} \sum_{j \in \mathcal{N}_\varepsilon(p_i)} \left(\|\tilde{n}_j^\pi \cdot (\tilde{p}_i^e - p_j^\pi)\|^2 + \frac{1}{|\mathcal{N}_\varepsilon(p_i)|} \|y_{i,1} \cdot (\tilde{p}_i^e - p_j^\pi)\|^2 \right). \tag{8.13}$$

The above energy function is defined on the plane H_i. In comparison to Equation (8.11), we include an additional summand. This is necessary, because the matrix of the linear system resulting from Equation (8.13) without the summand would not be invertible. The reason is, that the least dominant eigenvalue along the normal being zero would reduce the rank of the corresponding matrix. We choose this additional summand which is directed along the edge to create an orthonormal basis as the plane H_i is spanned using the other two eigenvectors of the weighted covariance matrix C_i. Including the summand, we can minimize Equation (8.13) once more by solving a linear system which results in the equation:

$$p_i'^e = \left(\sum_{j \in \mathcal{N}_\varepsilon(p_i)} \tilde{n}_j^\pi \otimes \tilde{n}_j^\pi + y_{i,1} \otimes y_{i,1} \right)^{-1} \sum_{j \in \mathcal{N}_\varepsilon(p_i)} \left(\tilde{n}_j^\pi \otimes \tilde{n}_j^\pi p_j + y_{i,1} \otimes y_{i,1} p_i^e \right),$$

where the summand $y_1 \otimes y_1$ ensures that the matrix is invertible. The term $p_i'^e$ is a temporary vertex position and we once more compute the variation between $p_i'^e$ and the corresponding original vertex from the noisy point set to modify the edge vertex accordingly:

$$\tilde{p}_i^e = \begin{cases} p_i'^e & \text{if} \quad v_i \leq \nu \\ p_i^e & \text{if} \quad v_i > \nu \end{cases}. \tag{8.14}$$

For each edge vertex, the above equation computes the optimal position by minimizing the distance between the lines, which are defined by the projected vertex normals. This operation effectively preserves sharp features along edges and removes noise effectively.

Vertex Update on Flat Regions Finally, let us consider a point $p_i^f \in P_f$, labeled as point within a planar area by Section 8.2.2. In this flat region, the matrix C_i has two dominant eigenvalues. In order to remove noise in these regions, we allow to move the vertex position only in direction of the corresponding vertex normal $\tilde{n}_i$. Thereby, we follow the approach of [Sun+07; Zhe+17]. We use an energy function similar to that of Equation (8.11), but with the restriction to only move in normal direction. That is, we find the updated position $\tilde{p}_i^f$ by minimizing

$$\min_{s \in \mathbb{R}} \sum_{p_j \in \mathcal{N}_\varepsilon(p_i)} W_{ij} |\langle \tilde{n}_j, p_i^f + s \cdot \tilde{n}_i \rangle - \langle \tilde{n}_j, p_j \rangle|^2. \tag{8.15}$$

Similar to edge and corner vertex updates, we first compute the variation v_i and then update the vertex position according to

$$\tilde{p}_i^f = \begin{cases} p_i^f + \frac{\alpha}{\sum_{j \in \mathcal{N}_\varepsilon(p_i)} W_{ij}} \sum_{j \in \mathcal{N}_\varepsilon(p_i)} W_{ij} \langle \tilde{n}_j, p_j - p_i^f \rangle \tilde{n}_i & \text{if} \quad v_i \leq \nu \\ p_i^f & \text{if} \quad v_i > \nu \end{cases}, \tag{8.16}$$

where α is a user-controlled parameter to limit the amount of smoothing on flat regions and W_{ij} is a combination of a similarity and a closeness function:

$$W_{ij} = \exp\Big(- \frac{16 |\tilde{n}_i - \tilde{n}_j|^2}{\delta^2} \Big) \cdot \exp\Big(- \frac{4 |p_j - p_i^f|^2}{\delta^2} \Big), \tag{8.17}$$

where δ is half the diameter of the point set $\mathcal{N}_\varepsilon(p_i)$.

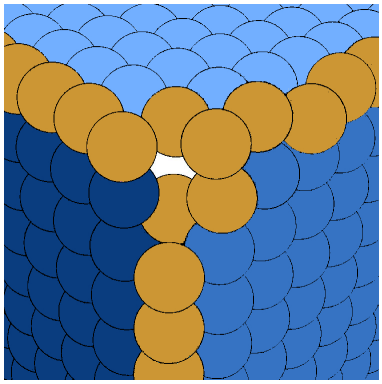
Even though the update scheme for flat regions as given in Equation (8.16) seems most elaborate, the combination with simpler schemes for both corners (8.12) and edges (8.14) is more effective in practice, as shown in Figure 8.5.

8.2.4 Method Summary

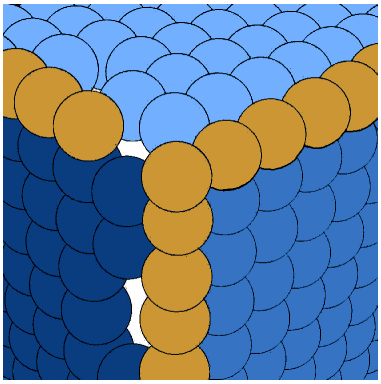
In the previous Sections 8.2.1, 8.2.2, and 8.2.3, we have presented the three key steps of our smoothing method. By iteratively applying these three steps to a noisy point set input, the proposed algorithm produces a noise-free point set with proper sharp features. The whole method is summarized in Algorithm 5.

8.3 Experimental Results

We evaluate the capacity of our algorithm on various kinds of point set models corrupted with synthetic noise (Figures 8.7, 8.8, 8.10, 8.12, 8.13) and real scanned data (Figure 8.14). We compared our method with five state-of-the-art denoising Methods [MC17], [Zhe+17], [GG07], [ÖGG09], and [Zhe+18]. Methods [GG07] and [ÖGG09] are implemented in MeshLab, see [Cig+08]. The results of the methods [MC17], [Zhe+17], and [Zhe+18] are provided by the authors.



(a) *The point set reconstructed by using Equation (8.16) not only for flat regions but also for feature points.*

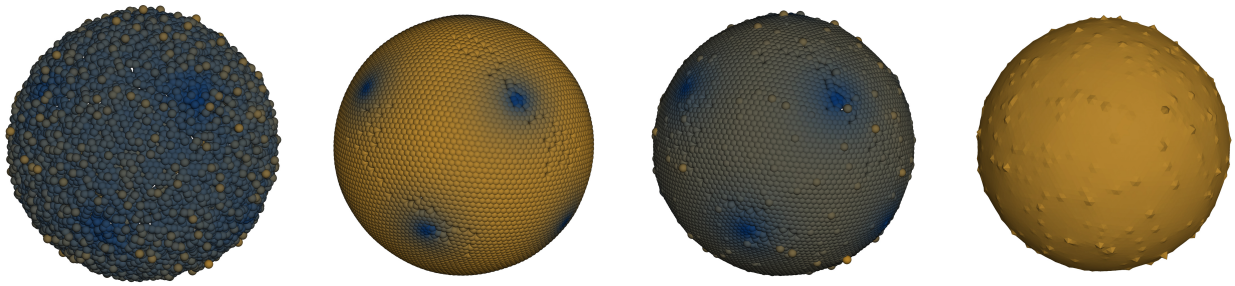


(b) *The result obtained by the proposed scheme, where flat regions follow Equation (8.16), edges are reconstructed using Equation (8.14) and corner positions are updated using Equation (8.12).*

Figure 8.5: *This figure shows the effect of the proposed constraint-based vertex position update scheme. Note how the corner itself is not recovered in (a), but is recovered in (b).*

Algorithm 5 Constraint-based Point Set Denoising using the Normal Voting Tensor and Restricted Quadratic Error Metric

procedure DENOISE(Point Set P, number of iterations I, radius ε, normal threshold ρ, feature threshold τ, flat region threshold α, corner threshold ν, damping factor D)
  for I iterations **do**
    for all vertex $p_i \in P$ **do** ▷ Vertex Normal Filtering
      Compute NVT T_i by (8.1)
      Perform BEO on the spectral components of T_i
      Compute altered NVT $\tilde{T}_i$ by (8.6)
    end for
    for all vertex $p_i \in P$ **do** ▷ Feature Detection
      Compute weighted covariance matrix C_i by (8.8)
      Partition P into three disjoint sets P_f, P_e, P_c representing face-, edge-, and corner-points by Section 8.2.2
    end for
    for all vertex $p_i \in P_c$ **do** ▷ Vertex Update Corners
      Apply constraint-based vertex update by (8.11)
    end for
    for all vertex $p_i \in P_e$ **do** ▷ Vertex Update Edges
      Apply constraint-based vertex update by (8.13)
    end for
    for all vertex $p_i \in P_f$ **do** ▷ Vertex Update Faces
      Apply constraint-based vertex update by (8.15)
    end for
  end for
end procedure



(a) $\sigma_n = 0.25\ell_a$ **(b)** $\nu = 2\ell_a$ **(c)** $\nu = \ell_a$ **(d)** $\nu = \ell_a$

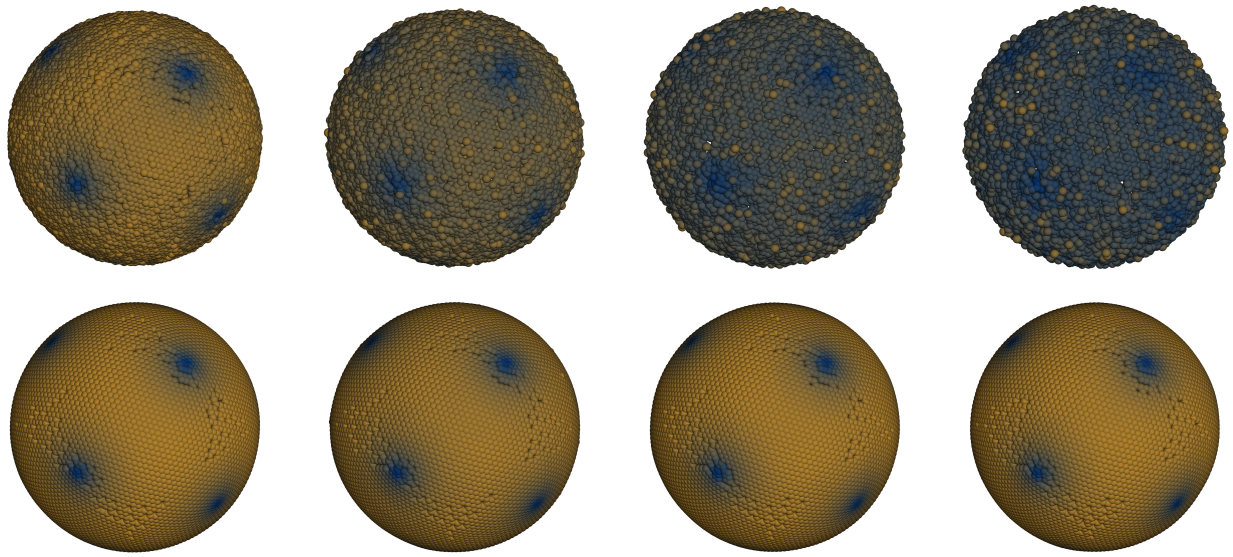
Figure 8.6: *Effect of the distance-based constraint ν. Figure (a) shows a sphere, which is corrupted by Gaussian noise in random direction with standard deviation $\sigma_n = 0.25\ell_a$, where ℓ_a is the average distance between vertices of the point set. Figure (b) represents a desirable noise-free output with $\nu = 2\ell_a$. Figures (c) and (d) show that when $\nu = \ell_a$, the proposed method is not able to remove all noise components. Here, Figure (d) is a triangulation of the point set shown in Figure (c) using the Ball Pivot algorithm, [Ber+99]. The values of E_v (8.18) are 0.984 and 0.981 for Figure (b) and (c) respectively. Vertices are colored based on the variation of the distance of each vertex to its neighbors.*

8.3.1 Parameter Tuning

We introduced several parameters: the geometric neighbor radius ε, the dihedral angle threshold ρ (Equation (8.2)), the eigenvalue threshold τ (Section 8.2.1), the damping factor D (Equation (8.7)), the distance-based constraint ν (Equations (8.12), (8.14), (8.16)), the total number of iterations I, and the vertex-based diffusion speed α (Equation (8.16)). Throughout the whole experimentation, we fix $\alpha = 0.1$ and $D = 3$. The radius ε of the geometric neighborhood depends on the resolution of the input point set and it is fixed to be twice the average distance ℓ_a between the vertices of the point set. The average distance between the vertices is computed using $k = 6$ nearest neighbors of each vertex. For these experiments, we also fix the distance-based constraint $\nu = 2\varepsilon$ (except Figure 8.6). Effectively, there are only 3 parameters (τ, ρ, I) to tune the results. In the quantitative comparison, see Table 8.1, the parameters are mentioned in the following format: (τ, ρ, I). For Methods [MC17], [Zhe+17], and [Zhe+18], we mention "Default" in the parameter column because the corresponding smooth models are provided by their authors. For the method [GG07], we used the parameters (h, $\#\{iterations\}$, α), and for [ÖGG09], we used (σ_r, σ_n), both listed in Table 8.1.

The eigenvalue threshold τ depends on the noise intensity on a point set. The bigger the noise intensity, the larger the value of τ should be chosen. We use $\tau \in \{0.25, \ldots, 0.4\}$ for synthetic data and $\tau \in \{0.05, \ldots, 0.1\}$ for real data because in our experiments, real data point sets have smaller noise intensity compared to synthetic data point sets. We iterate several times ($I \in \{20, \ldots, 100\}$) for best results. The term $\rho \in \{0.8, \ldots, 0.95\}$ is the threshold to select the neighbor components and it is computed using the scalar product between the neighbor vertex normals. On a CAD model, we choose a high threshold value because of sharp features and on CAGD models, we choose a small threshold value because of smoother features compared to CAD models. The distance-based constraint ν is one of the most important parameter in the proposed algorithm. The effect of this parameter is shown in Figure 8.6. As it can be seen, a small value of ν leads to less shrinkage (small E_v) but does not remove all noise components.

Note that there is a dependency amongst the different thresholds. We have stated above that τ needs to be smaller than the lowest eigenvalue of all T_i respectively. However, these depend on both ε and ρ. Therefore, to guess a suitable τ, one needs to compute T_i by Equation (8.1) for



(a) $\sigma_n = 0.13\ell_a$ **(b)** $\sigma_n = 0.25\ell_a$ **(c)** $\sigma_n = 0.35\ell_a$ **(d)** $\sigma_n = 0.5\ell_a$

Figure 8.7: *Shrinkage analysis during the denoising process. The sphere model is corrupted with different levels of noise in random directions (top) and the denoising process is applied (bottom). The measure E_v (8.18) is computed to quantify the shrinkage effect in the proposed method. From Figures (a)-(d), the values of E_v are 0.984, 0.983, 0.982, and 0.9811 respectively. Vertices are colored based on the variation of distances to their neighbors.*

all $i = 1, \ldots, n$, determining the smallest dominant eigenvalue. Only then, τ can be chosen in a meaningful way.

8.4 Quantitative Analysis

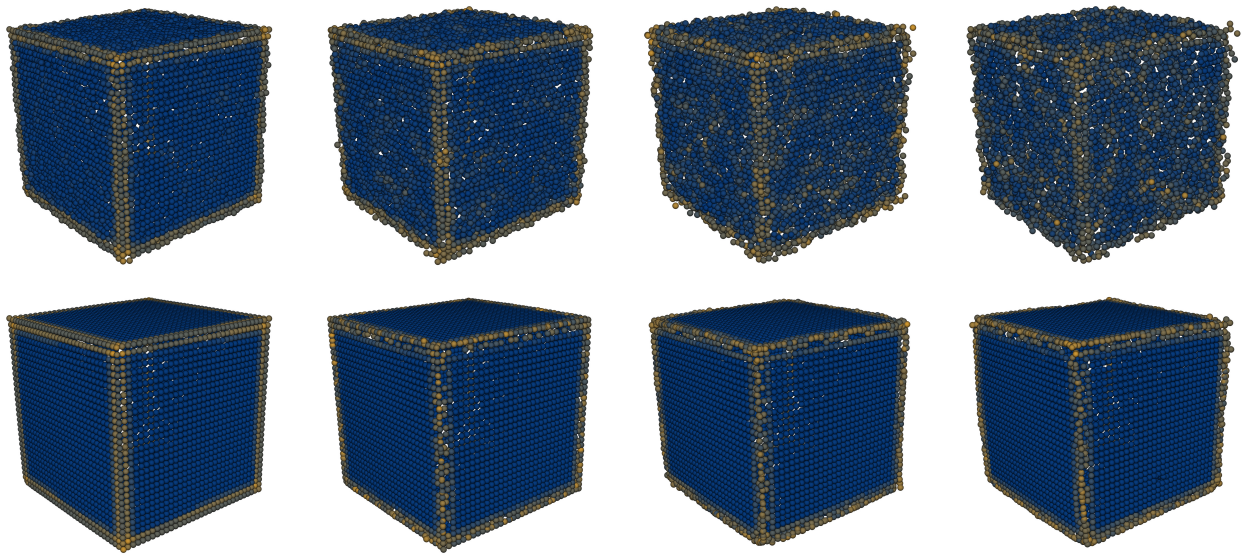
We performed several experiments regarding the quantitative analysis of the proposed algorithm. In general, shrinkage and feature blurring are two main challenges during the denoising process. In this section, we show the behavior of the proposed algorithm against different levels of noise in terms of shrinkage and feature preservation.

To quantify shrinkage, we performed the denoising process with a unit sphere and computed the following error measure:

$$E_v = \frac{1}{n} \sum_{i=0}^{n-1} \|p_i\|^2. \tag{8.18}$$

For the original Sphere $E_v = 1.0$ and due to shrinkage effects, the value of E_v decreases. As shown in Figure 8.7, the shrinkage effect increases with noise intensity but at the same time, these changes are not large. The value of E_v also depends on the distance-based constraint ν. As shown in Figure 8.6, with a bigger value of ν, it is possible that E_v will be bigger but at the same time it gives a smoother result compared to a small value of ν.

For further shrinkage analysis, we reconstructed triangulated surfaces from the denoised point sets using the "ball pivoting" algorithm [Ber+99] (Figures 8.10–8.14). To compute the closeness between the ground truth model and the denoised model, we use the following vertex-based error



(a) $\sigma_n = 0.13\ell_a$ **(b)** $\sigma_n = 0.25\ell_a$ **(c)** $\sigma_n = 0.35\ell_a$ **(d)** $\sigma_n = 0.5\ell_a$

Figure 8.8: *A visual representation of feature preservation analysis. The cube model is corrupted with different levels of noise in random directions. To measure the feature preservation capability of the proposed algorithm, we computed the MAD (8.20) and obtain values* 2.99, 4.15, 6.4, *and* 6.48 *for Figures (a)–(d) respectively. Vertices are colored based on the variation of vertex normals.*

measure, which is defined as in [Sun+07]:

$$D_v = \sqrt{\frac{1}{3\sum_{k\in F} a_k} \sum_{i\in V} \sum_{j\in F_v(i)} a_j \mathrm{dist}(\tilde{p}_i, T)^2} \tag{8.19}$$

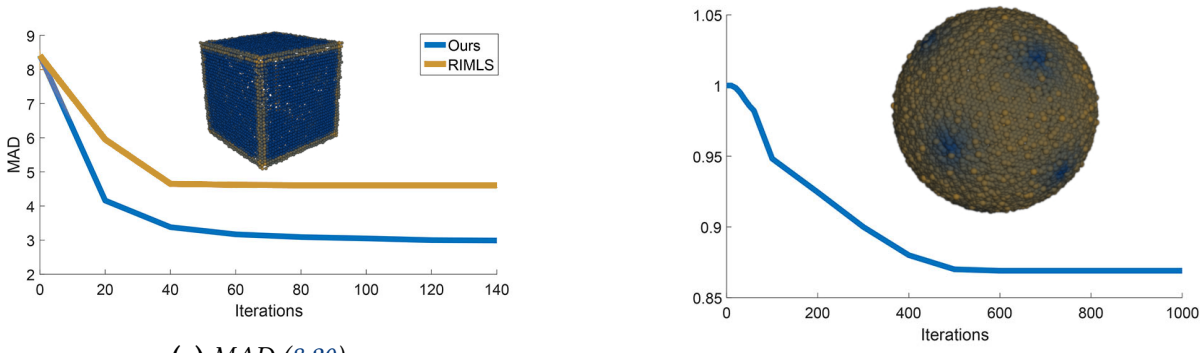
where F and V are treated as the triangular element set and the set of vertices respectively after the triangulated surface reconstruction. The terms a_k and a_j are the corresponding face areas. The distance $\mathrm{dist}(\tilde{p}_i, T)$ is the closest L_2-distance between the newly computed vertex $\tilde{p}_i$ and the triangle T of the reference model. Values of D_v for several models are given in Table 8.1.

To quantify feature preservation, we check the orientation error between the denoised model and the ground truth. *Mean angular deviation* (MAD) is defined to measure the orientation error:

$$MAD = \frac{1}{n} \sum_{i=0}^{n-1} \measuredangle(\bar{n}_i, \tilde{n}_i), \tag{8.20}$$

where $\bar{n}_i$ and $\tilde{n}_i$ are vertex normals of the ground truth model and the denoised model respectively. Figure 8.8 shows that the MAD is large when the noise intensity is high. So, with bigger noise, the orientation error will be bigger compared to lower noise. As it can be seen from Figure 8.8, for $\sigma_n = 0.13\ell_a$ to $\sigma_n = 0.35\ell_a$, the output models are noise-free with sharp features. However, with $\sigma_n = 0.5\ell_a$, we are not able to preserve all sharp features. Values of MAD for several models are given in Table 8.1.

Figure 8.9 shows the convergence property of the proposed algorithm, where Figure 8.9a shows the orientation error with a noisy cube model which is almost constant after about 100 iterations. As it can be seen from the figure, the proposed method has better convergence rate compared to RIMLS [ÖGG09]. Our method has improved convergence rate because of the BEO (see page 108)



(a) *MAD (8.20)*

(b) E_v *(8.18)*

Figure 8.9: *A visual representation of the convergence of the proposed algorithm. The left image shows the orientation error convergence on a cube model ($\sigma_n = 0.13\ell_a$) while the right image demonstrates the shrinkage measure E_v over iterations on a noisy sphere ($\sigma_n = 0.25\ell_a$). In both cases, ℓ_a denotes the average distance between vertices of the point set.*

which assigns binary values to the eigenvalues of the vertex-based NVT. By assigning zero to the least dominant eigenvalue, our algorithm removes noise components faster compared to state-of-the-art methods. Similarly, Figure 8.9b shows the variation of E_v w.r.t. iterations. As the number of iterations increases, the value of E_v decreases, which leads to shrinkage effects. In the proposed method, the shrinkage effect is controlled using the distance-based constraint ν. As it can be seen from Figure 8.9b, the value of E_v is almost constant after about 400 iterations because the value of ν is set to approximately twice of the point set resolution.

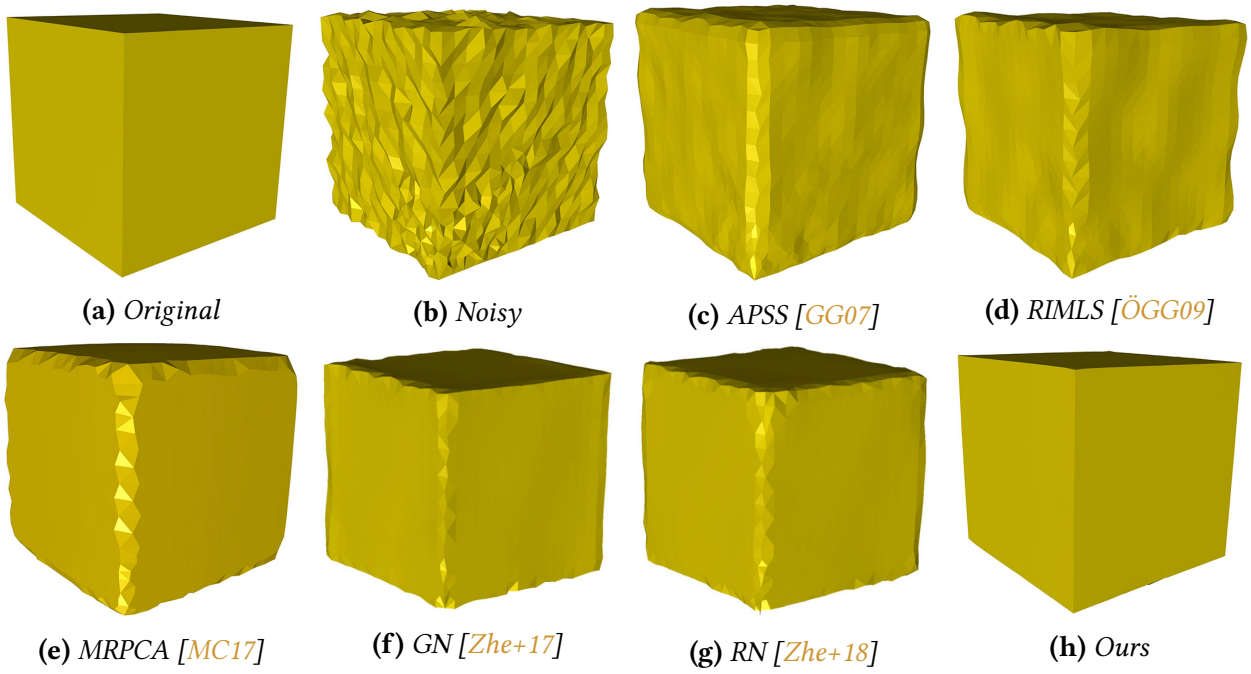
Table 8.1 shows the comparison of the proposed method with five state-of-the-art methods. As it can be seen, for the cube model shown in Figure 8.10, our method not only reconstructs sharp features (low MAD) but also produces minimum shrinkage (low D_v) compared to the current state-of-the-art methods. For the rocker arm model, the proposed method is not as good as APSS [GG07] or RIMLS [ÖGG09] in terms of MAD and D_v. However, Figure 8.11 shows that our method produces smoother umbilical regions with enhanced sharp features on this model. For the fandisk model given in Figure 8.12, the proposed algorithm performs better compared to state-of-the-art methods in terms of feature preservation. However, it produces more volume shrinkage compared to APSS [GG07]. Similar to the cube model, our algorithm outperforms state-of-the-art methods in terms of feature preservation and volume shrinkage. For a visual comparison with state-of-the-art methods, see the original publication as given in the list of publications prior to the thesis, page 5.

8.5 Conclusion

In this section, we presented a simple and effective tensor multiplication algorithm for feature-preserving point set denoising. The proposed method is an extension of the ENVT-based mesh denoising [YRP17]. Similar to the concept of the ENVT, in the proposed algorithm, we used vertex-based NVT and the spectral analysis of this tensor leads to decoupling of features from noise. Noise components are removed by the multiplication of the vertex-based NVT to the corresponding vertex normal. The concept of BEO not only enhances sharp features but also improves the convergence rate of the method. Local binary neighborhood selection helps to select similar vertices in the neighborhood to compute the vertex-based NVT to avoid feature blurring during the denoising process. After the vertex normal filtering, we classify feature points into edges,

Table 8.1: *Quantitative Comparison*

Models	Methods	MAD	$D_v \times 10^{-3}$	Parameters[3]
Cube $\lvert V\rvert = 1906$ Figure 8.10	[GG07]	5.56	3.24	(2, 45, 0.5)
	[ÖGG09]	4.62	5.41	(4, 0.75)
	[MC17]	4.60	3.37	Default
	[Zhe+17]	3.48	7.51	Default
	[Zhe+18]	4.47	6.46	Default
	Ours	**2.85**	**1.65**	(0.3, 0.95, 150)
Rocker arm $\lvert V\rvert = 24106$ Figure 8.11	[GG07]	**5.13**	22.6	(4, 15, 0.5)
	[ÖGG09]	5.26	**21.4**	(4, 1)
	[MC17]	6.31	33.0	Default
	[Zhe+17]	8.14	118.7	Default
	[Zhe+18]	6.26	72.12	Default
	Ours	7.56	43.26	(0.25, 0.9, 80)
Fandisk $\lvert V\rvert = 25894$ Figure 8.12	[GG07]	**3.72**	1.7	(4, 15, 0)
	[ÖGG09]	6.6	1.81	(4, 0.75)
	[MC17]	13.67	1.56	Default
	[Zhe+17]	4.57	1.81	Default
	[Zhe+18]	4.34	1.4	Default
	Ours	4.4	**1.39**	(0.3, 0.9, 150)
Octahedron $\lvert V\rvert = 40242$ Figure 8.13	[GG07]	3.35	0.27	(2, 45, 0.5)
	[ÖGG09]	4.31	0.39	(4, 0.75)
	[MC17]	4.6	0.32	Default
	[Zhe+17]	1.2	0.52	Default
	[Zhe+18]	1.37	0.49	Default
	Ours	**1.11**	**0.19**	(0.25, 0.9, 80)



(a) *Original* **(b)** *Noisy* **(c)** *APSS [GG07]* **(d)** *RIMLS [ÖGG09]*

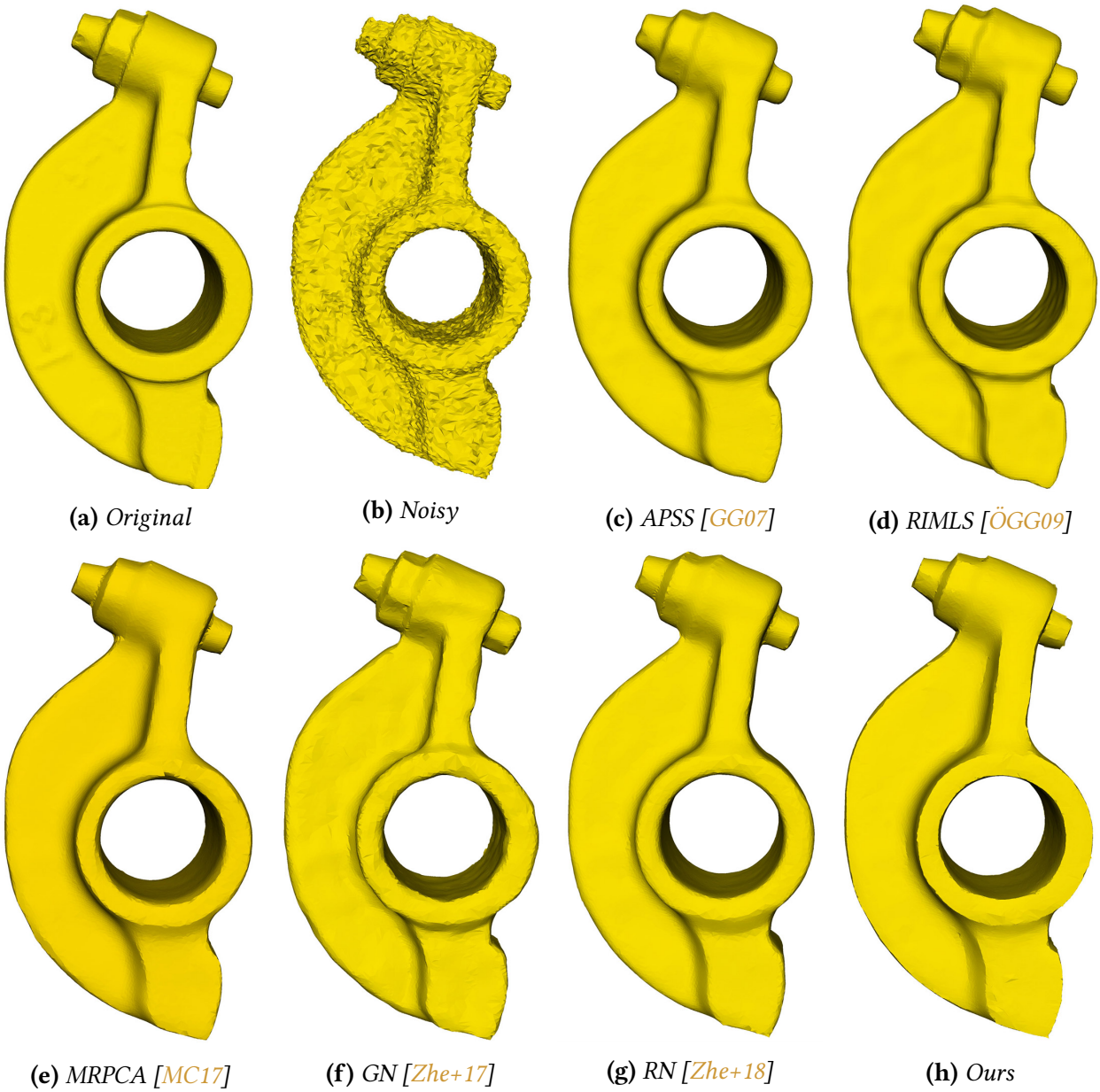
(e) *MRPCA [MC17]* **(f)** *GN [Zhe+17]* **(g)** *RN [Zhe+18]* **(h)** *Ours*

Figure 8.10: *The cube model with non-uniform distribution of vertices, corrupted by Gaussian noise ($\sigma_n = 0.3\ell_a$) in normal direction, where ℓ_a is the average distance between the vertices of the model. It can be seen that the proposed method is able to preserve sharp features effectively compared to state-of-the-art methods and does not create bumpy structures. Surfaces are reconstructed using the "ball pivoting" algorithm, [Ber+99].*

corners, and flat regions using an anisotropic covariance matrix. For the vertex update stage, we introduced restricted least square error metrics, which are different for different kinds of features. The vertex position reconstruction using restricted quadratic error metrics helps the algorithm to recreate the sharp edges and corners. The experimental results show the effectiveness of the proposed algorithm.

Our method is capable of handling noise, but yields erroneous results under high noise intensities. This is based on the fact, that noise has a great impact on the normal estimation and the NVT construction, which we use throughout the whole process. Another issue arises when the input point set is highly irregular. As can be seen in Figure 8.14, our method is robust up to a moderate level of irregularity but it is possible that with extreme irregular sampling, the output may not be satisfactory.

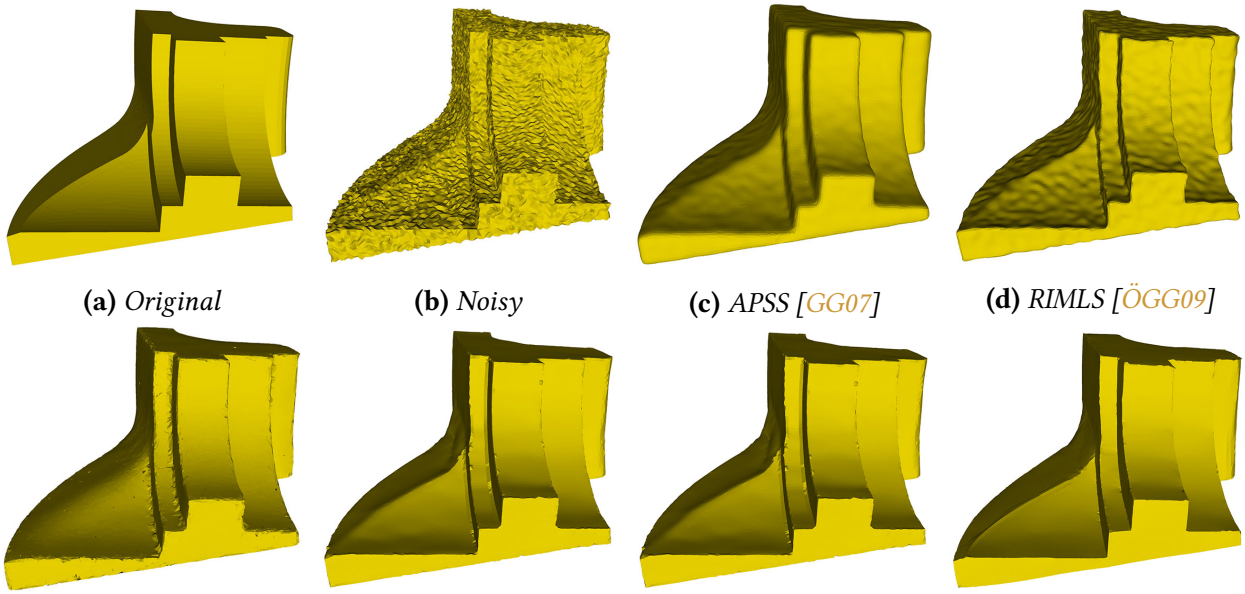
During the denoising process, we tuned the different parameters manually to get the desired results. We need to find an optimal combination of the parameters automatically. A direction which we have in part considered in Section 1.5.1, but will continue to investigate in the future.



(a) *Original* **(b)** *Noisy* **(c)** *APSS [GG07]* **(d)** *RIMLS [ÖGG09]*

(e) *MRPCA [MC17]* **(f)** *GN [Zhe+17]* **(g)** *RN [Zhe+18]* **(h)** *Ours*

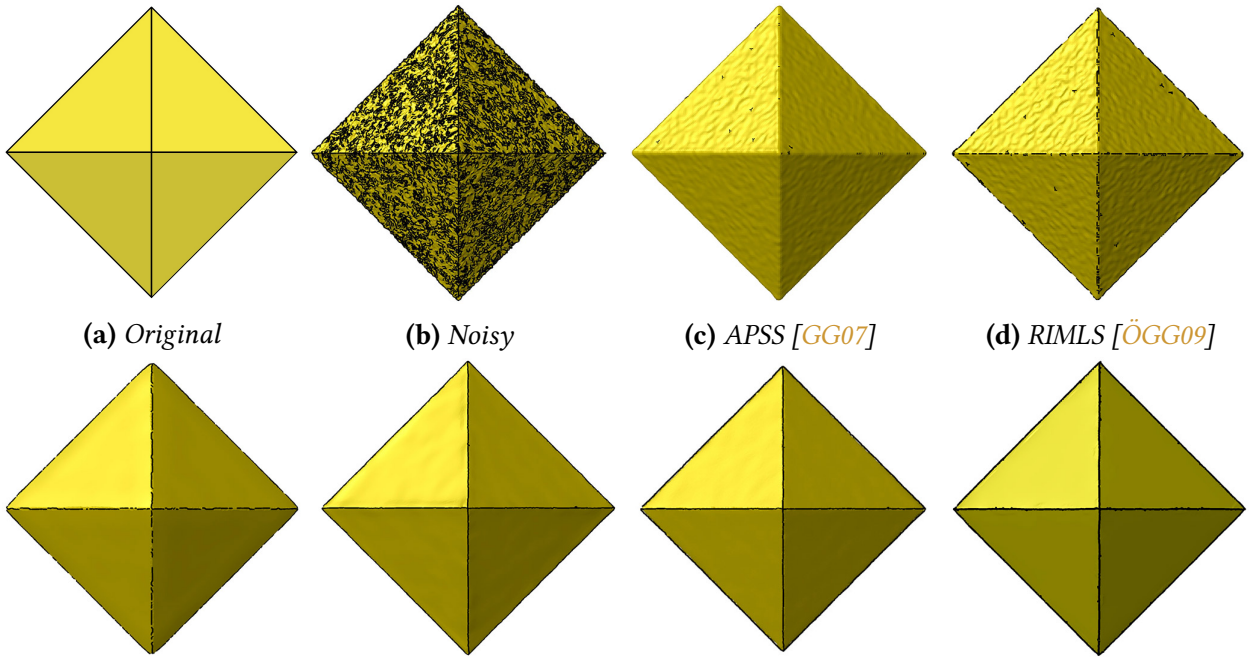
Figure 8.11: *The rocker arm model corrupted by Gaussian noise ($\sigma_n = 0.3\ell_a$) in normal direction, where ℓ_a is the average distance between the vertices of the model. The results are produced by state-of-the-art methods and our proposed method. The proposed method removes noise effectively and also enhances the sharp features around the cylindrical region. Surfaces are reconstructed using the "ball pivoting" algorithm, [Ber+99].*



(a) *Original* **(b)** *Noisy* **(c)** *APSS [GG07]* **(d)** *RIMLS [ÖGG09]*

(e) *MRPCA [MC17]* **(f)** *GN [Zhe+17]* **(g)** *RN [Zhe+18]* **(h)** *Ours*

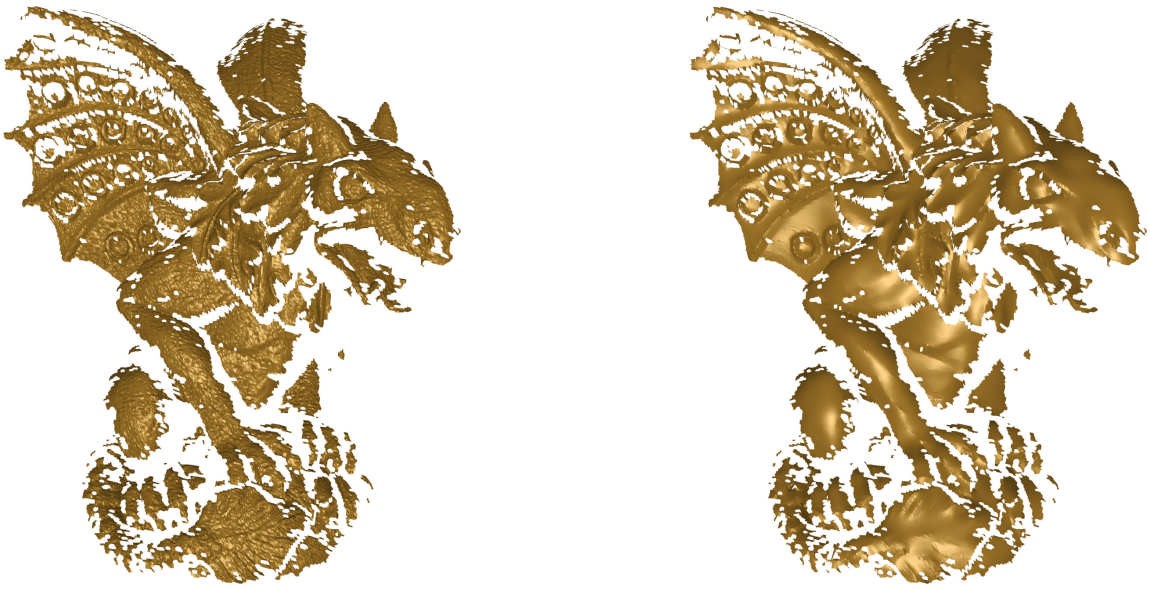
Figure 8.12: *The fandisk model corrupted by Gaussian noise ($\sigma_n = 0.28\ell_a$) in normal direction, where ℓ_a is the average distance between the vertices of the model. It can be seen that the proposed method is able to preserve sharp features effectively compared to state-of-the-art methods. Surfaces are reconstructed using the "ball pivoting" algorithm [Ber+99].*



(a) *Original* **(b)** *Noisy* **(c)** *APSS [GG07]* **(d)** *RIMLS [ÖGG09]*

(e) *MRPCA [MC17]* **(f)** *GN [Zhe+17]* **(g)** *RN [Zhe+18]* **(h)** *Ours*

Figure 8.13: *The octahedron model, which is corrupted by Gaussian noise in normal direction and results produced by the proposed method and state-of-the-art methods. Surfaces are reconstructed using the "ball pivoting" algorithm, [Ber+99].*



(a) *Noisy irregular real data.* **(b)** *Our result.*

Figure 8.14: *Robustness against irregular data points. Figure (a) shows the noisy irregular data points of the gargoyle model and Figure (b) shows the result obtained by the proposed method.*

Conclusion and Further Research

Throughout the thesis, we have motivated the importance of point set processing for a multitude of applications. Consequently, we have explored three different topics revolving around both the theory and the practical usage of point sets.

Notions of Neighborhood and corresponding Data Structures The first topic concerned notions of neighborhood and corresponding data structures. We introduced a new anisotropic neighborhood concept which takes the local shape of the geometry into account. That is, the neighborhoods are built using normal information of the point set, see Section 1.3. We evaluated our new concept both in several experiments (Section 1.4) and by including it into two different applications: Section 1.5.1 showed how point set denoising can benefit from the proposed neighborhoods, while we also used them to construct a shape-aware version of the Moving Least Squares (MLS) procedure in Section 1.5.2. While the latter compares well with the Robust Implicit Moving Least Squares (RIMLS) of [ÖGG09] visually, a thorough evaluation is left for further research.

In any practical application, the fast determination of neighborhoods is important. In other words, no neighborhood concept can be used in practice if it does not come with an algorithm for computing it efficiently. A prominent choice for these computations is the k-d tree data structure of [FBF77]. Its main achievement is a proven expected runtime of $\mathcal{O}(\log(n))$ for every neighborhood query on n points. To make its underlying theory more accessible, the very concise original proof has been reworked and elaborated in Section 2 of this thesis.

As a final aspect of the first chapter, we noted that the data structure of k-d trees is not designed for parallel execution. As current workstations can delegate computations to the graphics card and thus profit from parallelization, this is an aspect of growing importance. Hence, we investigated a corresponding data structure in Section 3, namely the neighborhood grid of [Jos+09; Jos+15]. Despite a thorough study of [MW15], several open questions remained of which some have been solved this thesis:

- a proof of asymptotic time-optimality of a comparison-based building algorithm (Theorem 5),
- several combinatorial results on the number of possible sorted placements in the grid (Section 3.2),
- a complete list of unique sorted placements for $n \in [3]$ as well as a proof for non-existence of unique sorted placements for $n \geq 4$ (Section 3.3),
- results on the neighborhood quality obtained from the data structure (Section 3.6).

The questions for a specific point set attaining the least or largest number of stable states for $n \geq 4$ and for a complete theoretical evaluation of the neighborhood quality are left for further research, while we presented a conjecture (see Conjecture 1) for the upper bound of number of stable states.

Manifold Structure for Point Set Surfaces The second large topic of this thesis dealt with manifold structures for point set surfaces. When the underlying real-world geometry has the structure of a surface manifold, it can be expected that this structure is reflected by the point set acquired from the geometry. However, there was no theory available to establish what is meant by a manifold point set surface. The thesis first established that regarding a point set as 0-manifold is not a very practical choice, see Section 4.2. Therefore, we restricted to the setting of point sets sampling some d'-manifold in $\mathbb{R}^d$, $d > d'$. For this setup, we presented a scheme by which the point set can be treated as manifold utilizing a transition manifold, see Section 4.3. While many algorithms could be used for the computation of the transition manifold, we exemplified our theoretical scheme with the MLS procedure of [SL16].

As manifolds are a collection of charts, the fast and practical generation of charts for point sets is the main necessary step for creating point set manifolds. While the aforementioned MLS procedure of [SL16] creates very localized charts, the approach of [Li+11] parametrizes the point set as a whole and thus gives one very large coordinate representation. The thesis aimed for a solution between these two extremes and thus turned to the Variational Shape Approximation (VSA) of [CAD04]. We generalized the procedure to the setting of point sets and enriched it with a split and merge step to become independent of the number of charts sought. Furthermore, we gave a technique to obtain a simplified surface mesh from the segmented point set. See Section 5 for a complete discussion. The extension of our surface simplification procedure to non-star-convex surface patches is left for further research.

Robust and Efficient Processing of Point Sets In the third and final chapter of the thesis, we elaborated on the fact that algorithms have to work efficiently and robustly on the given input point set. While meshed geometries provide an intuitive and natural weighting by the areas of the faces, point sets can at most work with distances between the points. This makes in particular the handling of non-uniform point set samples difficult. Starting from an idea originally presented in [LP05], we defined a discrete directional density measure for point sets. This measure can be computed completely intrinsically without information aside from the input point set. From the measure, we deduced a set of weights for the usage e.g. in discretized differential-geometry operators to make them robust against non-uniform density sampling. Aside from these theoretical results, Section 6 includes several empirical studies to show the superiority of the presented weights. Other applications—like the implementation of our weights into the contexts of point set registration, simplification, or surface reconstruction—are left for further research.

The real-world objects represented by point sets will in practical applications not all be smooth, but include sharp features like edges and corners. The detection of these is important e.g. in the context of denoising (see below). Even though there are several feature detection algorithms available for point sets, none of them comes with any mathematical guarantee to detect a certain range of features. We employed quantities derived from the MLS procedure of [SL16] to benefit from the corresponding theory (see Theorem 13). In Section 7, we presented the different quantities and evaluate the detection in several experiments.

Finally, during the acquisition of the point set, noise can be introduced. Removing it while retaining the sharp features (see above) of the underlying geometry is a challenging task. In the final Section 8 of the thesis, we translated an algorithm of [YRP17] to the context of point sets. This gave an iterative smoothing algorithm. In order to make it robust in the denoising of corners, we introduced quadratic error metrics for the vertex update stage in the smoothing pipeline. A thorough comparison to state-of-the-art algorithms proved the proposed method to be superior on a wide range of examples.

Appendices

A Notation

absolute value	$\mathrm{abs}(x) := \lvert x \rvert, x \in \mathbb{R}$
average vertex distance	ℓ_a
binary neighborhood threshold	$\rho \in \mathbb{R}_{>0}$
BEO threshold	$\tau \in \mathbb{R}_{>0}$
covariance matrix	C_i
damping factor	$D \in \mathbb{R}_{>0}$
degree (of polynomial)	$m \in \mathbb{N}_0$
density (local)	δ_i
diffusion speed	$\alpha \in \mathbb{R}_{>0}$
dimension	$d \in \mathbb{N}$
lower dimension	$d' \in \mathbb{N}, d' < d$
disjoint union	$A \dot\cup B$
distance measure	$d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}, (p, q) \mapsto d(p, q)$
eigenvalues of covariance matrix	$\lambda_{i,\ell}, \ell = 1, \ldots, d, \lambda_{i,1} \geq \ldots \geq \lambda_{i,d}$
eigenvectors of the covariance matrix	$v_{i,\ell}, \ell \in [d]$ (corresponding to the above decreasing sorting of the eigenvalues)
eigenvalues of NVT	$\lambda_{i,\ell}, \ell = 1, \ldots, d, \lambda_{i,1} \geq \ldots \geq \lambda_{i,d}$
eigenvectors of NVT	$x_{i,\ell}, \ell \in [d]$ (corresponding to the above decreasing sorting of the eigenvalues)
feature value	$\mu^{\beth}, \beth \in \{2rad, double, dist, Func\}$
Landau symbols	upper bound $\mathcal{O}(n)$, upper and lower bound $\Theta(n)$, lower bound $\Omega(n)$
logarithm	$\log(x)$, to basis 2 if not indicated otherwise
manifold (smooth), (approximated)	$\mathcal{M}, \tilde{\mathcal{M}}$
norm	$\lVert p \rVert$ denotes the L^2 norm $\lVert . \rVert_2$, if not indicated otherwise
natural numbers	$\mathbb{N} = \{0, 1, 2, \ldots\}$
neighborhood (combinatorial)	$\mathcal{N}_k(p_i)$
neighborhood (metric)	$\mathcal{N}_\varepsilon(p_i)$
neighborhood (both)	$\mathcal{N}_{k,\varepsilon}$
normal at a point p_i	$n_i \in \mathbb{R}^d$
normal field on a point set P	N_p
number of neighbors	$k \in \mathbb{N}$
number of points	$n \in \mathbb{N}$
scalar product	$\langle p_i, p_j \rangle$ denotes the Euclidean scalar product for points $p_i, p_j \in \mathbb{R}^d$ if not indicated otherwise
squarenumber of points	$N = n^2$
point	$p_i = (p_i^1, \ldots, p_i^d)^T$
point set	$P = \{p_i \in \mathbb{R}^d \mid i = 1, \ldots, n\}$
positive real numbers	$\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}$
radius for neighbors	$\varepsilon \in \mathbb{R}_{\geq 0}$
radius for weight function θ	$r \in \mathbb{R}_{\geq 0}$
range of natural numbers	$[n] = \{1, \ldots, n\}, n \in \mathbb{N}$
transpose	$p^T \in \mathbb{R}^{d \times 1}$
weight function	$\theta : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$

B Statistical Experiment Results for the Shape-Aware Neighborhoods

In this appendix, we present the full statistical data from the experiments presented in Section 1.4. These experiments consist of computing the values $E_i^{\dim}$ (1.11) and E_i^{λ} (1.12) with modifications discussed in Section 1.3 for each point p_i of the considered point set P. The point sets used in our experiments are the noiseless CAD models "bearing", Figure B.1, and "fandisk", Figure B.2, and the real-world models: "bunny", Figure B.3, and "kitten", Figure B.4.

In Tables B.1–B.4, we give the statistics on the measures E^{dim} and E^{λ}. Each cell contains (from top to bottom):

- the minimum $\min_{i=1}^{n} E_i^{dim}$,
- the maximum $\max_{i=1}^{n} E_i^{dim}$,
- the average as defined in (1.15),
- and the standard deviation of the value $E_i^{\dim}$

over all points p_i of the respective point set. For each model there are two tables showing in the left one the E^{dim}-values and in the right table the E^{λ}-values. In each table the lowest average and standard deviation value is marked in yellow. Note that some entries might be equal and apply for the minimum, but this happens due to rounding processes.

Some of the cells are stating "n.a.". This happens when at least one point reports empty neighborhoods for all $k \in \{6, \dots, 20\}$. That means, we do not find any point with a normal similar enough to the one of the considered point for the given parameters a and b in the sigmoid weight detection. Hence we leave out the error calculation for both error metrics, as this would mislead in the comparison. But nevertheless, these values indicate, that we have to increase the collection rate or choose a smaller comparison rate and increase of the sigmoid.

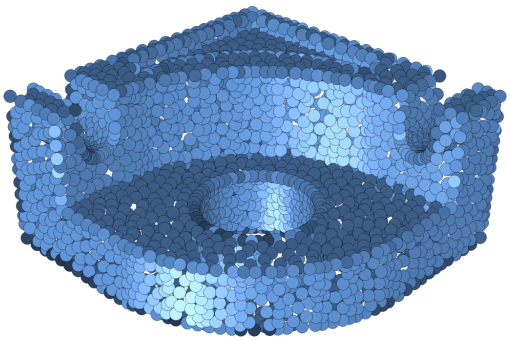


Figure B.1: *bearing model (3,475 vertices)*

	$E^{\dim}$			
a \ b	1	2	4	∞
0.0	0.0177	0.0177	0.0177	0.0177
	0.9857	0.9849	0.9823	1.0062
	0.4876	0.4997	0.5011	0.5021
	0.1957	0.2008	0.2019	0.2027
0.25	0.0177	0.0177	0.0177	0.0177
	0.9501	0.976	0.9711	0.9823
	0.4747	0.4941	0.4981	0.4985
	0.1893	0.1985	0.1997	0.2001
0.5	0.0177	0.0177	0.0177	0.0177
	0.9603	0.9445	0.9464	0.9629
	0.4481	0.4581	0.4605	0.4801
	0.1765	0.183	0.184	0.1937
0.75	0.0177	0.0177	0.0177	0.0177
	0.9142	0.9704	0.9392	0.9392
	0.4209	0.4408	0.4472	0.4514
	0.1593	0.1711	0.1757	0.1777
0.9	n.a.	n.a.	n.a.	n.a.

	E^{λ}			
a \ b	1	2	4	∞
0.0	0.1922	0.1924	0.1924	0.1924
	0.9799	0.9817	0.9632	1.0165
	0.6388	0.6465	0.6478	0.6498
	0.0821	0.0872	0.0891	0.0933
0.25	0.1805	0.1924	0.1924	0.1924
	0.9694	0.9685	0.9685	0.9685
	0.6312	0.6427	0.6452	0.6456
	0.0785	0.0841	0.0859	0.086
0.5	0.1381	0.1778	0.1818	0.1818
	0.9619	0.9614	0.9685	0.9685
	0.6152	0.6203	0.6219	0.6349
	0.078	0.076	0.0755	0.0827
0.75	0.0783	0.0771	0.0771	0.0771
	0.8574	0.8935	0.8824	0.9198
	0.5923	0.605	0.6098	0.6128
	0.0912	0.0842	0.0823	0.0809
0.9	n.a.	n.a.	n.a.	n.a.

Table B.1: *Results obtained from the bearing model with 3,475 vertices, see Figure B.1. On the left values for $E^{\dim}$ (1.11) and on the right values for E^{λ} (1.12). Each cell showing (from top to bottom) the minimum, maximum, average, and standard deviation.*

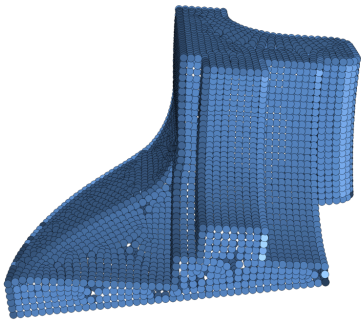


Figure B.2: *fandisk model (6,475 vertices)*

	$E^{\dim}$			
a \ b	1	2	4	∞
0.0	4.0E-4	4.0E-4	4.0E-4	4.0E-4
	0.9993	0.9676	0.9676	0.9676
	0.3464	0.3548	0.3549	0.3549
	0.2177	0.2262	0.2262	0.2262
0.25	4.0E-4	4.0E-4	4.0E-4	4.0E-4
	0.9914	0.9979	0.9676	0.9676
	0.338	0.3523	0.3548	0.3548
	0.2135	0.2237	0.2262	0.2262
0.5	4.0E-4	4.0E-4	4.0E-4	4.0E-4
	0.9984	0.9722	0.9714	0.9676
	0.3263	0.3254	0.3267	0.3455
	0.205	0.2138	0.2145	0.222
0.75	4.0E-4	4.0E-4	4.0E-4	4.0E-4
	0.9546	0.9707	0.9705	0.964
	0.3113	0.3284	0.3266	0.3248
	0.1744	0.2079	0.2112	0.2131
0.9	0.0	0.0	0.0	0.0
	0.7611	0.8516	0.8806	0.9021
	0.2776	0.2787	0.2827	0.2869
	0.138	0.1392	0.1451	0.1509

	E^{λ}			
a \ b	1	2	4	∞
0.0	0.4282	0.4282	0.4282	0.4282
	0.9149	0.9236	0.9236	0.9236
	0.6503	0.6527	0.6527	0.6527
	0.0698	0.0747	0.0747	0.0747
0.25	0.4282	0.4282	0.4282	0.4282
	0.9077	0.9236	0.9236	0.9236
	0.648	0.6518	0.6527	0.6527
	0.0659	0.0729	0.0747	0.0747
0.5	0.4282	0.4282	0.4282	0.4282
	0.8973	0.9232	0.9236	0.9236
	0.6436	0.6448	0.6451	0.6498
	0.0604	0.0631	0.0634	0.0705
0.75	0.2289	0.3185	0.3259	0.3259
	0.8741	0.8977	0.894	0.897
	0.6334	0.6421	0.6435	0.6441
	0.0489	0.0602	0.0618	0.0625
0.9	0.0	0.0	0.0	0.0
	0.7468	0.7606	0.7855	0.8335
	0.5875	0.5909	0.5932	0.5962
	0.1021	0.0981	0.0959	0.0929

Table B.2: *Results obtained from the fandisk model with 6,475 vertices, see Figure B.2. On the left values for $E^{\dim}$ (1.11) and on the right values for E^{λ} (1.12). Each cell showing (from top to bottom) the minimum, maximum, average, and standard deviation.*

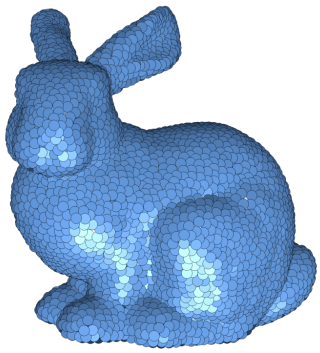


Figure B.3: *bunny model (4,899 vertices)*

	$E^{\dim}$			
a \ b	1	2	4	∞
0.0	0.0534	0.0534	0.0534	0.0534
	0.8973	0.9025	0.9159	0.9822
	0.3897	0.3919	0.3927	0.3937
	0.1574	0.1585	0.1597	0.1615
0.25	0.0532	0.0534	0.0534	0.0534
	0.8967	0.9004	0.9041	0.9041
	0.3884	0.3905	0.3909	0.3916
	0.1562	0.1566	0.1574	0.1582
0.5	0.0491	0.0534	0.0534	0.0534
	0.8984	0.896	0.896	0.8978
	0.387	0.3879	0.3888	0.3895
	0.1556	0.1546	0.1548	0.1554
0.75	0.0474	0.0534	0.0534	0.0484
	0.8416	0.9207	0.8895	0.896
	0.3853	0.386	0.3867	0.3873
	0.1545	0.1539	0.154	0.1545
0.9	0.0	0.0	0.0	0.0
	0.8332	0.824	0.8306	0.8603
	0.3863	0.3827	0.3835	0.384
	0.1541	0.1529	0.1527	0.1527

	E^{λ}			
a \ b	1	2	4	∞
0.0	0.1918	0.2068	0.2068	0.2068
	0.8783	0.8719	0.873	1.0053
	0.6568	0.6582	0.6588	0.6595
	0.0489	0.0496	0.0501	0.0524
0.25	0.1803	0.2014	0.2068	0.2068
	0.8833	0.8719	0.8719	0.873
	0.6554	0.6568	0.6574	0.658
	0.0489	0.0492	0.0496	0.0497
0.5	0.161	0.172	0.1724	0.1923
	0.8962	0.8719	0.8719	0.8719
	0.6533	0.6548	0.6553	0.6558
	0.0496	0.0495	0.0494	0.0492
0.75	0.104	0.1458	0.1691	0.1719
	0.8515	0.8789	0.9062	0.8714
	0.6469	0.6499	0.651	0.6518
	0.0554	0.0526	0.0517	0.0511
0.9	0.0	0.0	0.0	0.0
	0.8352	0.8314	0.8314	0.8314
	0.6208	0.6293	0.6331	0.6361
	0.0921	0.0842	0.0794	0.0752

Table B.3: *Results obtained from the bunny model with 4,899 vertices, see Figure B.3. On the left values for $E^{\dim}$ (1.11) and on the right values for E^{λ} (1.12). Each cell showing (from top to bottom) the minimum, maximum, average, and standard deviation.*

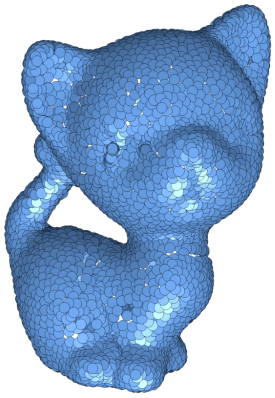


Figure B.4: *kitten model (5,011 vertices)*

	$E^{\dim}$			
a \ b	1	2	4	∞
0.0	0.0306	0.0312	0.0312	0.0312
	0.8823	0.8976	0.9425	0.8976
	0.4074	0.4083	0.4084	0.4086
	0.1543	0.1548	0.1549	0.155
0.25	0.0301	0.0312	0.0312	0.0312
	0.8767	0.8976	0.8976	0.8976
	0.4068	0.4078	0.4081	0.4082
	0.1542	0.1543	0.1546	0.1547
0.5	0.0288	0.0312	0.0312	0.0312
	0.8737	0.8884	0.8976	0.8976
	0.4064	0.407	0.4074	0.4076
	0.1538	0.1538	0.1539	0.1542
0.75	0.0	0.0	0.0	0.0
	0.892	0.9125	1.0003	0.8478
	0.404	0.404	0.4046	0.4054
	0.1526	0.1523	0.1528	0.1531
0.9	n.a.	n.a.	n.a.	n.a.

	E^{λ}			
a \ b	1	2	4	∞
0.0	0.1748	0.1775	0.1775	0.1775
	0.8834	0.9135	0.9135	0.9135
	0.6175	0.6182	0.6183	0.6185
	0.0641	0.0641	0.0641	0.0641
0.25	0.1727	0.1682	0.1755	0.1775
	0.8873	0.9094	0.9135	0.9135
	0.6169	0.6178	0.618	0.6181
	0.0645	0.0642	0.0644	0.0644
0.5	0.1643	0.1682	0.1652	0.1652
	0.9009	0.8769	0.8718	0.8718
	0.6154	0.6169	0.6172	0.6175
	0.0657	0.0647	0.0645	0.0644
0.75	0.0	0.0	0.0	0.0
	0.8359	0.8501	0.9005	0.8867
	0.6088	0.6116	0.6129	0.614
	0.0745	0.0718	0.0702	0.0687
0.9	n.a.	n.a.	n.a.	n.a.

Table B.4: *Results obtained from the kitten model with 5,011 vertices, see Figure B.4. On the left values for $E^{\dim}$ (1.11) and on the right values for E^{λ} (1.12). Each cell showing (from top to bottom) the minimum, maximum, average, and standard deviation.*

C Beta Distribution

The proof in the Section 2.2 makes use of the Beta Distribution which we will present here briefly with the facts used in the proof above. We follow the description in [Geo15, Section 2.5.3]. It describes the waiting time for the k-th event of n events happening in $(0,1) \subset \mathbb{R}$, called $1, \dots, n$. Hence, the space of results as well as the sample space is $\Omega = (0,1)^n$. For all $1 \leq i \leq n$, $\omega \in (\omega_1, \dots, \omega_n) \in \Omega$, let $T_i(\omega) = \omega_i$ be the time of event i. Assume all ω_i are uniform distributed in $(0,1)$.

Now order the $T_i(\omega)$ strictly, which is possible since $\mathbb{P}(\bigcup_{i \neq j} \{\omega_i = \omega_j\}) = 0$, i.e. no two events happen at the same time. Call the order

$$T_{1:n} < \dots < T_{n:n} \text{ with } \{T_{1:n}, \dots, T_{n:n}\} = \{T_1(\omega), \dots, T_n(\omega)\},$$

then $T_{k:n}$ is the time of the k-th event. Now fix k, n, and some $c \in (0,1)$. For a given order $T_{1:n} < \dots < T_{n:n}$, we obtain

$$\underbrace{\mathbb{P}(T_{k:n} \leq c)}_{\text{fixed order}} = \int_0^1 \dots \int_0^1 \mathbb{1}_{\{t_1 < \dots < t_n\}}(t_1, \dots, t_n) \cdot \mathbb{1}_{(0,c]}(t_k)\, dt_n \dots dt_1.$$

For any other order, we can exchange the integration by Fubini, s.t. it yields the same expression independent of any of the $n!$ possible orders. Hence,

$$\begin{aligned}
\mathbb{P}(T_{k:n} \leq c) &= n! \int_0^1 \dots \int_0^1 \mathbb{1}_{\{t_1 < \dots < t_n\}}(t_1, \dots, t_n) \cdot \mathbb{1}_{(0,c]}(t_k)\, dt_n \dots dt_1 \\
&= n! \cdot \underbrace{\int_0^1 \mathbb{1}_{(0,c]}(t_k)}_{= \int_0^c 1} \\
&\quad \cdot \Bigg[\underbrace{\left(\int_0^1 \dots \int_0^1 \mathbb{1}_{\{t_1 < \dots < t_k\}}(t_1, \dots, t_k)\, dt_{k-1} \dots dt_1 \right)}_{\int_0^{t_k} \dots \int_0^{t_k} \mathbb{1}_{\{t_1 < \dots < t_k\}}(t_1, \dots, t_{k-1})\, dt_{k-1} \dots dt_1} \\
&\quad \cdot \underbrace{\left(\int_0^1 \dots \int_0^1 \mathbb{1}_{\{t_k < \dots < t_n\}}(t_k, \dots, t_n)\, dt_{k+1} \dots dt_n \right)}_{= \int_{t_k}^1 \dots \int_{t_k}^1 \mathbb{1}_{\{t_{k+1} < \dots < t_n\}}(t_{k+1}, \dots, t_n)\, dt_n \dots dt_{k+1}} \Bigg]
\end{aligned} \tag{C.1}$$

Using the following two identities, see [Geo15, p. 44],

$$\begin{aligned}
\int_0^s \dots \int_0^s \mathbb{1}_{\{t_1 < \dots < t_{k-1}\}}(t_1, \dots, t_{k-1})\, dt_{k-1} \dots dt_1 &= \frac{s^{k-1}}{(k-1)!} \\
\int_0^s \dots \int_0^s \mathbb{1}_{\{t_{k+1} < \dots < t_n\}}(t_{k+1}, \dots, t_n)\, dt_n \dots dt_{k+1} &= \frac{(1-s)^{n-k}}{(n-k)!}
\end{aligned}$$

in (C.1), we get

$$\mathbb{P}(T_{k:n} \leq c) = \frac{n!}{(k-1)!(n-k)!} \cdot \int_0^c s^{k-1}(1-s)^{n-k}\, ds.$$

In general, for a random variable X with distribution F_X and density f, if F_X is differentiable, we have $f(x) = F'(X)$. Finally, it can be shown, see [Geo15, p. 45], that $\mathbb{E}(\beta_{a,b}) = \frac{a}{a+b}$, where for $k, n \in \mathbb{N}$, $\beta_{k,n-k+1}$ describes the distribution of the k-th smallest time of n random times in $[0,1]$. In particular, $\beta_{1,n}(s) = n(1-s)^{n-1}$ is the density of the first time and $\beta_{1,1} = \mathcal{U}_{(0,1)}$.

D Densities from Covariance Matrix

We now have a closer look on the expression of $\delta_{i,1}, \delta_{i,2}$, given by the two largest eigenvalues $\lambda_{i,1}, \lambda_{i,2}$ from (6.4), cf. [Tau95]. A unit direction on T_i can be expressed in terms of the orthonormal basis of the eigenvectors of the covariance matrix $v_{i,1}, v_{i,2}$. Hence,

$$e_\varphi = \begin{pmatrix} \cos(\varphi) \\ \sin(\varphi) \end{pmatrix}, \qquad \text{with} \qquad e_\varphi e_\varphi^T = \begin{pmatrix} \cos^2(\varphi) & \cos(\varphi)\sin(\varphi) \\ \cos(\varphi)\sin(\varphi) & \sin^2(\varphi) \end{pmatrix}. \tag{D.1}$$

for some $\varphi \in [0, 2\pi)$. We now compute the integral form of the tangential part of the diagonalized covariance matrix by using the quadratic form (6.3) and the matrix from (D.1)

$$\begin{aligned}
\begin{pmatrix} \lambda_{i,1} & 0 \\ 0 & \lambda_{i,2} \end{pmatrix} =& \frac{1}{2\pi} \int_0^{2\pi} \delta(e_\varphi) e_\varphi e_\varphi^T \, d\varphi \\
=& \frac{1}{2\pi} \int_0^{2\pi} \delta(e_\varphi) \begin{pmatrix} \cos^2(\varphi) & \cos(\varphi)\sin(\varphi) \\ \cos(\varphi)\sin(\varphi) & \sin^2(\varphi) \end{pmatrix} d\varphi \\
=& \frac{1}{2\pi} \delta_{i,1} \int_0^{2\pi} \begin{pmatrix} \cos^4(\varphi) & \cos^3(\varphi)\sin(\varphi) \\ \cos^3(\varphi)\sin(\varphi) & \cos^2(\varphi)\sin^2(\varphi) \end{pmatrix} d\varphi \\
&+ \frac{1}{2\pi} \delta_{i,2} \int_0^{2\pi} \begin{pmatrix} \cos^2(\varphi)\sin^2(\varphi) & \cos(\varphi)\sin^3(\varphi) \\ \cos(\varphi)\sin^3(\varphi) & \sin^4(\varphi) \end{pmatrix} d\varphi \\
=& \frac{1}{2\pi} \delta_{i,1} \begin{pmatrix} \frac{3\pi}{4} & 0 \\ 0 & \frac{\pi}{4} \end{pmatrix} + \frac{1}{2\pi} \delta_{i,2} \begin{pmatrix} \frac{\pi}{4} & 0 \\ 0 & \frac{3\pi}{4} \end{pmatrix} \\
=& \delta_{i,1} \begin{pmatrix} \frac{3}{8} & 0 \\ 0 & \frac{1}{8} \end{pmatrix} + \delta_{i,2} \begin{pmatrix} \frac{1}{8} & 0 \\ 0 & \frac{3}{8} \end{pmatrix}.
\end{aligned}$$

This leaves us with the two equations

$$\lambda_{i,1} = \frac{3}{8}\delta_{i,1} + \frac{1}{8}\delta_{i,2} \qquad \lambda_{i,2} = \frac{1}{8}\delta_{i,1} + \frac{3}{8}\delta_{i,2},$$

which are rearranged to express $\delta_{i,1}, \delta_{i,2}$ in terms of the eigenvalues

$$\delta_{i,1} = 3\lambda_{i,1} - \lambda_{i,2} \qquad \delta_{i,2} = -\lambda_{i,1} + 3\lambda_{i,2}$$

to finally obtain equation (6.5).

Bibliography

[ACK01] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. "The Power Crust". In: *Proceedings of the sixth ACM symposium on Solid modeling and applications*. ACM. 2001, pp. 249–266.

[AK04] Nina Amenta and Yong Joo Kil. "Defining Point-set Surfaces". In: *ACM Trans. Graph.* 23.3 (Aug. 2004), pp. 264–270. ISSN: 0730-0301.

[AKS83] Miklós Ajtai, János Komlós, and Endre Szemerédi. "Sorting in c logn parallel steps". In: *Combinatorica* 3.1 (1983), pp. 1–19.

[Ale+01] Marc Alexa et al. "Point Set Surfaces". In: *VIS'01: Proceedings of the confernce on Visualization '01*. IEEE Computer Society. 2001, pp. 21–28.

[Ale+03] Marc Alexa et al. "Computing and rendering point set surfaces". In: *IEEE Transactions on visualization and computer graphics* 9.1 (2003), pp. 3–15.

[Ale+05] Marc Alexa et al., eds. *Eurographics Symposium on Point-Based Graphics (2005)*. 2005.

[Ama+96] Nancy M. Amato et al. "A comparison of parallel sorting algorithms on different architectures". In: *Technical Report TR98-029, Department of Computer Science, Texas A&M University* (1996).

[And+98] Arne Andersson et al. "Sorting in linear time?" In: *Journal of Computer and System Sciences* 57.1 (1998), pp. 74–93.

[AP10] Marco Attene and Giuseppe Patanè. "Hierarchical structure recovery of point-sampled surfaces". In: *Computer Graphics Forum*. Vol. 29. 6. Wiley Online Library. 2010, pp. 1905–1920.

[Att+06] Marco Attene et al. "Mesh Segmentation – A Comparative Study". In: *Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006*. 2006, pp. 7–18.

[Avr+10] Haim Avron et al. "L_1-Sparse Reconstruction of Sharp Point Set Surfaces". In: *ACM Trans. Graph.* 29.5 (Nov. 2010), 135:1–135:12. ISSN: 0730-0301.

[Bel+14] Ben Bellekens et al. "A survey of rigid 3D pointcloud registration algorithms". In: *AMBIENT 2014: the Fourth International Conference on Ambient Computing, Applications, Services and Technologies, August 24-28, 2014, Rome, Italy*. 2014, pp. 8–13.

[Ben75] Jon Louis Bentley. "Multidimensional Binary Search Trees Used for Associative Searching". In: *Communications of the ACM* 18.9 (1975), pp. 509–517.

[Ber+00] Marc de Berg et al. *Computational Geometry*. 2nd. Springer, 2000.

[Ber+17] Matthew Berger et al. "A Survey of Surface Reconstruction from Point Clouds". In: *Comput. Graph. Forum* 36.1 (Jan. 2017), pp. 301–329. ISSN: 0167-7055.

[Ber+99] Fausto Bernardini et al. "The ball-pivoting algorithm for surface reconstruction". In: *IEEE transactions on visualization and computer graphics* 5.4 (1999), pp. 349–359.

[BF05] Chris Boehnen and Patrick Flynn. "Accuracy of 3D Scanning Technologies in a Face Scanning Scenario". In: *IEEE Fifth International Conference on 3D Digital Imaging and Modeling*. 2005, pp. 310–317.

[BH14] Joseph K Blitzstein and Jessica Hwang. *Introduction to probability*. Chapman and Hall/CRC, 2014.

[Bis06] Christopher Michael Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[BJ88] Paul J. Besl and Ramesh C. Jain. "Segmentation Through Variable-Order Surface Fitting". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 10 (2) (1988), pp. 167–192.

[BL06] David Belton and Derek D. Lichti. "Classification and segmentation of terrestrial laser scanner point clouds using local variance information". In: *The International Archives of the Photogrammetry, Remote Sensing, and Spatial Information Sciences* 36.5 (2006), pp. 44–49.

[BL12] Nicolas Brodu and Dimitri Lague. "3D terrestrial lidar data classification of complex natural scenes using a multi-scale dimensionality criterion: Applications in geomorphology". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 68 (2012), pp. 121–134.

[Blu+73] Manuel Blum et al. "Time Bounds for Selection". In: *Journal of Computer and System Sciences* 7.4 (1973), pp. 448–461.

[Bot+06] Mario Botsch et al., eds. *Symposium on Point-Based Graphics*. 2006.

[Bot+07] Mario Botsch et al., eds. *Eurographics Symposium on Point-Based Graphics*. 2007.

[Bri07] Robert Bridson. "Fast Poisson disk sampling in arbitrary dimensions". In: *SIGGRAPH sketches*. 2007, p. 22.

[BSW09] Mikhail Belkin, Jian Sun, and Yusu Wang. "Constructing Laplace operator from Point Clouds in $\mathbb{R}^d$". In: *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2009, pp. 1031–1040.

[Buc+07] Ursula Buck et al. "Application of 3D documentation and geometric reconstruction methods in traffic accident analysis: With high resolution surface scanning, radiological MSCT/MRI scanning and real data based animation". In: *Forensic science international* 170.1 (2007), pp. 20–28.

[BZK09] David Bommes, Henrik Zimmer, and Leif Kobbelt. "Mixed-Integer Quadrangulation". In: *ACM Transactions On Graphics (TOG)*. Vol. 28. 3. ACM. 2009, p. 77.

[CAD04] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. "Variational Shape Approximation". In: *ACM Transactions on Graphics (TOG)*. Vol. 23. 3. ACM. 2004, pp. 905–914.

[CGF09] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. "A benchmark for 3D mesh segmentation". In: *Acm transactions on graphics (tog)*. Vol. 28. 3. ACM. 2009, 73:1–73:12.

[Che+13] Jiazhou Chen et al. "Non-Oriented MLS Gradient Fields". In: *Computer Graphics Forum* 32.8 (2013), pp. 98–109. ISSN: 1467-8659.

[Che+17] Siheng Chen et al. "Fast Resampling of 3D Point Clouds via Graphs". In: *arXiv preprint arXiv:1702.06397* (2017).

[Cig+08] Paolo Cignoni et al. "MeshLab: an Open-Source Mesh Processing Tool". In: *Eurographics Italian Chapter Conference*. Ed. by Vittorio Scarano, Rosario De Chiara, and Ugo Erra. The Eurographics Association, 2008, pp. 129–136. ISBN: 978-3-905673-68-5.

[CLT14] Louis Cuel, Jacques-Olivier Lachaud, and Boris Thibert. "Voronoi-based geometry estimator for 3D digital surfaces". In: *International Conference on Discrete Geometry for Computer Imagery*. Springer. 2014, pp. 134–149.

[CRS98] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. "Metro: measuring error on simplified surfaces". In: *Computer Graphics Forum*. Vol. 17 (2). Wiley Online Library. 1998, pp. 167–174.

[Cue+15] Louis Cuel et al. "Robust Geometry Estimation using the Generalized Voronoi Covariance Measure". In: *SIAM Journal on Imaging Sciences* 8.2 (2015), pp. 1293–1314.

[Cue14] Louis Cuel. "Inférence géométrique discrète". Thèse de doctorat dirigée par Lachaud, Jacques-Olivier et Thibert, Boris, Mathématiques appliquées Grenoble 2014. PhD thesis. École doctorale mathématiques, sciences et technologies de l'information, informatique (Grenoble), 2014.

[Dem+07] Kris Demarsin et al. "Detection of closed sharp edges in point clouds using normal estimation and graph theory". In: *Computer-Aided Design* 39.4 (2007), pp. 276–283.

[Dem+11] Jerome Demantké et al. "Dimensionality based scale selection in 3D lidar point clouds". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 38.Part 5 (2011), 97–102.

[Fah+14] Adil Fahad et al. "A Survey of Clustering Algorithms for Big Data: Taxonomy and Empirical Analysis". In: *Emerging Topics in Computing, IEEE Transactions on* 2.3 (2014), pp. 267–279.

[FBF77] Jerome Harold Friedman, Jon Louis Bentley, and Raphael Ari Finkel. "An algorithm for finding best matches in logarithmic expected time". In: *ACM Transactions on Mathematical Software (TOMS)* 3.3 (1977), pp. 209–226.

[FCS05] Shachar Fleishman, Daniel Cohen-Or, and Cláudio T Silva. "Robust Moving Least-squares Fitting with Sharp Features". In: *ACM transactions on graphics (TOG)* 24.3 (2005), pp. 544–552.

[FDC03] Shachar Fleishman, Iddo Drori, and Daniel Cohen-Or. "Bilateral Mesh Denoising". In: *ACM Trans. Graph.* 22.3 (July 2003), pp. 950–953. ISSN: 0730-0301.

[FH73] Keinosuke Fukunaga and Larry D. Hostetler. "Optimization of k nearest neighbor density estimates". In: *IEEE Transactions on Information Theory* 19.3 (1973), pp. 320–326.

[FHT01] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. Springer series in statistics Springer, Berlin, 2001.

[FR01] Michael S Floater and Martin Reimers. "Meshless parametrization and surface reconstruction". In: *Computer Aided Geometric Design* 18.2 (2001), pp. 77–92.

[Geo15] Hans-Otto Georgii. *Stochastik: Einführung in die Wahrscheinlichkeitstheorie und Statistik*. Walter de Gruyter GmbH & Co KG, 2015.

[GF16] Iliyan Georgiev and Marcos Fajardo. "Blue-noise dithered sampling". In: *ACM SIGGRAPH 2016 Talks*. ACM. 2016, p. 35.

[GG07] Gaël Guennebaud and Markus Gross. "Algebraic Point Set Surfaces". In: *ACM Trans. Graph.* 26.3 (July 2007). ISSN: 0730-0301.

[GG99] Volker Gaede and Oliver Günther. "Multidimensional Access Methods". In: *ACM Computing Surveys* (July 1999), pp. 170–231.

[GH97] Michael Garland and Paul S. Heckbert. "Surface Simplification Using Quadric Error Metrics". In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. 1997, pp. 209–216.

[Gro+04] Markus Gross et al., eds. *SPBG'04 Symposium on Point - Based Graphics 2004*. 2004.

[GWM01] Stefan Gumhold, Xinlong Wang, and Rob MacLeod. "Feature Extraction from Point Clouds". In: *Proceedings of 10th International Meshing Round Table*. 2001, pp. 293–305.

[Hab72] A. Nico Habermann. *Parallel neighbor-sort (or the glory of the induction principle)*. Tech. rep. Carnegie-Mellon University, 1972.

[HB14] Dirk Holz and Sven Behnke. "Registration of Non-Uniform Density 3D Point Clouds using Approximate Surface Reconstruction". In: *ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of*. VDE. 2014, pp. 1–7.

[HBC11] Paul Harris, Chris Brunsdon, and Martin Charlton. "Geographically weighted principal components analysis". In: *International Journal of Geographical Information Science* 25.10 (2011), pp. 1717–1736.

[HHL15] Zhen Hua, Zilong Huang, and Jinjiang Li. "Mesh Simplification Using Vertex Clustering Based on Principal Curvature". In: *International Journal of Multimedia and Ubiquitous Engineering* 10.9 (2015), pp. 99–110.

[HK06] Alexander Hornung and Leif Kobbelt. "Robust Reconstruction of Watertight 3D Models from Non-uniformly Sampled Point Clouds Without Normal Information". In: *Symposium on geometry processing*. 2006, pp. 41–50.

[Hop+92] Hugues Hoppe et al. "Surface reconstruction from unorganized points". In: *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*. ACM. 1992, pp. 71–78.

[Hua+13] Hui Huang et al. "Edge-Aware Point Set Resampling". In: *ACM Trans. Graph.* 32.1 (Feb. 2013), 9:1–9:12. ISSN: 0730-0301.

[Jak+15] Wenzel Jakob et al. "Instant field-aligned meshes." In: *ACM Trans. Graph.* 34.6 (2015), pp. 189–1.

[Jan17] Johanna Jansen. "Anisotropic Smoothing and Feature Detection of Large Point Clouds Using Principal Curvatures". In Cooperation with Carmeq GmbH, Berlin, Germany. MA thesis. Berlin, Germany: Freie Universität Berlin, 2017.

[JDZ04] Thouis R. Jones, Fredo Durand, and Matthias Zwicker. "Normal improvement for point rendering". In: *IEEE Computer Graphics and Applications* 24.4 (2004), pp. 53–56.

[Joh+14] Mallory M. Johnston et al. *3D printing in Zero-G ISS technology demonstration*. Tech. rep. NASA, 2014.

[Jos+09] Mark Joselli et al. "A Neighborhood Grid Data Structure for Massive 3D Crowd Simulation on GPU". In: *2009 VIII Brazilian Symposium on Games and Digital Entertainment (SBGAMES)*. IEEE. 2009, pp. 121–131.

[Jos+15] Mark Joselli et al. "Neighborhood grid: A novel data structure for fluids animation with GPU computing". In: *Journal of Parallel and Distributed Computing* 75 (2015), pp. 20–28.

[KCL09] Hyun Soo Kim, Han Kyun Choi, and Kwan H Lee. "Feature detection of triangular meshes based on tensor voting theory". In: *Computer-Aided Design* 41.1 (2009), pp. 47–58.

[KNP07] Felix Kälberer, Matthias Nieser, and Konrad Polthier. "QuadCover - Surface Parameterization using Branched Coverings". In: *Comput. Graph. Forum* 26.3 (2007), pp. 375–384.

[KS00] Klaus Köster and Michael Spann. "MIR: an approach to robust clustering-application to range image segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22 (5) (2000), pp. 430–444.

[LB16] Kai Wah Lee and Pengbo Bo. "Feature curve extraction from point clouds via developable strip intersection". In: *Journal of Computational Design and Engineering* 3.2 (2016), pp. 102–111.

[LCL06] Yaron Lipman, Daniel Cohen-Or, and David Levin. "Error Bounds and Optimal Neighborhoods for MLS Approximation". In: *Proceedings of the fourth Eurographics symposium on Geometry processing*. Eurographics Association. 2006, pp. 71–80.

[Lee12] John Lee. *Introduction to Smooth Manifolds*. 2nd Edition. Springer, 2012.

[Lei85] Tom Leighton. "Tight bounds on the complexity of parallel sorting". In: *IEEE Transactions on Computers* 100.4 (1985), pp. 344–354.

[Lev+00] Marc Levoy et al. "The Digital Michelangelo Project: 3D Scanning of Large Statues". In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 2000, pp. 131–144.

[Lev04] David Levin. "Mesh-Independent Surface Interpolation". In: *Geometric modeling for scientific visualization*. Springer, 2004, pp. 37–49.

[Lev98] David Levin. "The approximation power of moving least-squares". In: *Mathematics of Computation of the American Mathematical Society* 67.224 (1998), pp. 1517–1531.

[Li+11] Er Li et al. "Meshless quadrangulation by global parameterization". In: *Computers & Graphics* 35.5 (2011), pp. 992–1000.

[Lin01] Lars Linsen. *Point cloud representation*. Tech. rep. Universität Karlsruhe, Faculty of Computer Science, 2001.

[Lip+07] Yaron Lipman et al. "Parameterization-free Projection for Geometry Reconstruction". In: *ACM Trans. Graph.* 26.3 (July 2007), pp. 22–27. ISSN: 0730-0301.

[Llo82] Stuart P. Lloyd. "Least squares quantization in PCM". In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137.

[LP01] Lars Linsen and Hartmut Prautzsch. "Local Versus Global Triangulations". In: *Proceedings of EUROGRAPHICS*. Vol. 1. 2001, pp. 257–263.

[LP05] Carsten Lange and Konrad Polthier. "Anisotropic Smoothing of Point Sets". In: *Computer Aided Design* 22 (2005), pp. 680–692.

[LPZ13] Jian Liang, Frederick Park, and Hongkai Zhao. "Robust and Efficient Implicit Surface Reconstruction for Point Clouds Based on Convexified Image Segmentation". In: *Journal of Scientific Computing* 54.2-3 (2013), pp. 577–602.

[LW85] Marc Levoy and Turner Whitted. *The Use of Points as a Display Primitive*. University of North Carolina, Department of Computer Science, 1985.

[Mar+06] Matthew R Marler et al. "The sigmoidally transformed cosine curve: a mathematical model for circadian rhythms with symmetric non-sinusoidal shapes". In: *Statistics in medicine* 25.22 (2006), pp. 3893–3904.

[MC17] Enrico Mattei and Alexey Castrodad. "Point cloud denoising via moving rpca". In: *Computer Graphics Forum*. Vol. 36. 8. Wiley Online Library. 2017, pp. 123–137.

[McL76] Dermot H. McLain. "Two dimensional interpolation from random data". In: *The Computer Journal* 19.2 (1976), pp. 178–181.

[MNG03] Niloy J. Mitra, An Nguyen, and Leonidas Guibas. "Estimating Surface Normals in Noisy Point Cloud Data". In: *International Journal of Computational Geometry & Applications* 14.04n05 (2003), pp. 261–276.

[Mos+17] Christian Mostegel et al. "Scalable Surface Reconstruction from Point Clouds with Extreme Scale and Density Diversity". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017, pp. 904–913.

[MPS19] Carmelo Mineo, Stephen Gareth Pierce, and Rahul Summan. "Novel algorithms for 3D surface point cloud boundary detection and edge reconstruction". In: *Journal of Computational Design and Engineering* 6.1 (2019), pp. 81–91.

[MTL00] Gérard Medioni, Chi-Keung Tang, and Mi-Suen Lee. "Tensor voting: Theory and applications". In: *Proceedings of RFIA, Paris, France* 3 (2000).

[Mul+14] Wolfgang Mulzer et al. "Approximate k-flat Nearest Neighbor Search". In: *arXiv preprint arXiv:1411.1519* (2014).

[MW15] Marcelo de Gomensoro Malheiros and Marcelo Walter. "Simple and Efficient Approximate Nearest Neighbor Search using Spatial Sorting". In: *28th SIBGRAPI Conference on Graphics, Patterns and Images*. IEEE. 2015, pp. 180–187.

[MWP18] Claudio Mura, Gregory Wyss, and Renato Pajarola. "Robust normal estimation in unstructured 3D point clouds by selective normal space exploration". In: *The Visual Computer* 34.6 (2018), pp. 961–971.

[Nea04] Andrew Nealen. *An As-Short-As-Possible Introduction to the Least Squares, Weighted Least Squares and Moving Least Squares Methods for Scattered Data Approximation and Interpolation*. Tech. rep. TU Darmstadt, Germany, 2004.

[NL13] Anh Nguyen and Bac Le. "3D point cloud segmentation: A survey". In: *2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM)*. 2013, pp. 225–230.

[ÖAG10] A. Cengiz Öztireli, Marc Alexa, and Markus Gross. "Spectral Sampling of Manifolds". In: *ACM Transactions on Graphics (TOG)* 29.6 (2010), 168:1–168:8.

[ÖGG09] A Cengiz Öztireli, Gael Guennebaud, and Markus Gross. "Feature preserving point set surfaces based on non-linear kernel regression". In: *Computer Graphics Forum*. Vol. 28. 2. Wiley Online Library. 2009, pp. 493–501.

[Oht+03] Yutaka Ohtake et al. "Multi-level Partition of Unity Implicits". In: *ACM Trans. Graph.* 22.3 (July 2003), pp. 463–470. ISSN: 0730-0301.

[Pau+03] Mark Pauly et al. "Shape modeling with point-sampled geometry". In: *ACM Transactions on Graphics* 22, 3 (2003), pp. 641–650.

[PGK02] Mark Pauly, Markus Gross, and Leif Kobbelt. "Efficient simplification of point-sampled surfaces". In: *Proceedings of the conference on Visualization'02*. IEEE Computer Society. 2002, pp. 163–170.

[PKG03] Mark Pauly, Richard Keiser, and Markus Gross. "Multi-scale Feature Extraction on Point-Sampled Surfaces". In: *Computer graphics forum*. Vol. 22. 3. Wiley Online Library. 2003, pp. 281–289.

[PKG06] Mark Pauly, Leif Kobbelt, and Markus Gross. "Point-based multiscale surface representation". In: *ACM Transactions on Graphics (TOG)* 25.2 (2006), pp. 177–193.

[PLL12] Min Ki Park, Seung Joo Lee, and Kwan H Lee. "Multi-scale tensor voting for feature extraction from unstructured point clouds". In: *Graphical Models* 74.4 (2012), pp. 197–208.

[PM92] Ola Petersson and Alistair Moffat. "A framework for adaptive sorting". In: *Algorithm Theory — SWAT '92*. Ed. by Otto Nurmi and Esko Ukkonen. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 422–433. ISBN: 978-3-540-47275-9.

[RT09] Dennie Reniers and Alexandru Telea. "Extreme simplification and rendering of point sets using algebraic multigrid". In: *Computing and Visualization in Science* 12 (1) (2009), pp. 9–22.

[Sag01] Bruce Eli Sagan. *The symmetric Group*. Springer, 2001.

[SAL17] B. Sober, Y. Aizenbud, and D. Levin. "Approximation of Functions over Manifolds: A Moving Least-Squares Approach". In: *ArXiv e-prints* (Nov. 2017).

[SFC10] Batchimeg Sosorbaram, Tadahiro Fujimoto, and Norishige Chiba. "Simplification of Point Set Surfaces using Bilateral Filter and Multi-Sized Splats". In: *The Journal of the Society for Art and Science* 9 (3) (2010), pp. 140–153.

[Sha48] C. E. Shannon. "A Mathematical Theory of Communication". In: *The Bell System Technical Journal* 27 (1948), pp. 379–423.

[Skr14b] Martin Skrodzki. "Neighborhood Computation of Point Set Surfaces". MA thesis. Freie Universität Berlin, 2014.

[SL16] Barak Sober and David Levin. "Manifold Approximation by Moving Least-Squares Projection (MMLS)". In: *ArXiv e-prints* (Aug. 2016).

[Sor+18] Tommaso Sorgente et al. "Topology-driven shape chartification". In: *Computer Aided Geometric Design* 65 (2018), pp. 13–28.

[SSW15] Yujing Sun, Scott Schaefer, and Wenping Wang. "Denoising Point Sets via L_0 Minimization". In: *Comput. Aided Geom. Des.* 35 (May 2015), pp. 2–15. ISSN: 0167-8396.

[Sun+07] Xianfang Sun et al. "Fast and Effective Feature-Preserving Mesh Denoising". In: *IEEE transactions on visualization and computer graphics* 13.5 (2007), pp. 925–938.

[SW11] Robert Sedgewick and Kevin Wayne. *Algorithms*. 4th. Addison-Wesley, 2011.

[SWK07] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. "Efficient RANSAC for point-cloud shape detection". In: *Computer graphics forum*. Vol. 26. 2. Wiley Online Library. 2007, pp. 214–226.

[Tau95] Gabriel Taubin. "Estimating the tensor of curvature of a surface from a polyhedral approximation". In: *Computer Vision, 1995. Proceedings., Fifth International Conference on*. IEEE. 1995, pp. 902–907.

[TP05] Daniel Tóvári and Norbert Pfeifer. "Segmentation based robust interpolation - A new Approach to laser data filtering". In: *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*. Vol. 36 (3/19). 2005, pp. 79–84.

[Wan+13] Jun Wang et al. "Feature-Preserving Surface Reconstruction From Unoriented, Noisy Point Data". In: *Computer Graphics Forum* 32.1 (2013), pp. 164–176. ISSN: 1467-8659.

[WHH11] Christopher Weber, Stefanie Hahmann, and Hans Hagen. "Methods for Feature Detection in Point Clouds". In: *OASIcs-OpenAccess Series in Informatics*. Vol. 19. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2011, pp. 90–99.

[WI18] Reed M. Williams and Horea T. Ilieş. "Practical shape analysis and segmentation methods for point cloud models". In: *Computer Aided Geometric Design* 67 (2018), pp. 97–120.

[WJM14] Martin Weinmann, Boris Jutzi, and Clément Mallet. "Semantic 3D Scene Interpretation: A Framework Combining Optimal Neighborhood Size Selection with Relevant Features". In: *ISPRS annals* II-3 (2014). ISPRS Technical Commission III Symposium, 5 – 7 September 2014, Zürich, CH, pp. 181–188. ISSN: 2194-9050.

[WW11] Li-Yi Wei and Rui Wang. "Differential domain analysis for non-uniform sampling". In: *ACM Transactions on Graphics (TOG)* 30.4 (2011), 50:1–50:10.

[XM09] Guo Xianglin and Pang Mingyong. "Point sets simplification using local surface analysis". In: *2009 2nd IEEE International Conference on Broadband Network Multimedia Technology*. 2009, pp. 575–579.

[Yad+18b] Sunil Kumar Yadav et al. "Constraint-based point set denoising using normal voting tensor and restricted quadratic error metrics". In: *Computers & Graphics* 74 (2018), pp. 234–243. ISSN: 0097-8493.

[Yan+17] Jiaqi Yang et al. "3D Reconstruction from Non-uniform Point Clouds via Local Hierarchical Clustering". In: *Ninth International Conference on Digital Image Processing (ICDIP 2017)*. Vol. 10420(38). International Society for Optics and Photonics. 2017.

[YRP17] Sunil Kumar Yadav, Ulrich Reitebuch, and Konrad Polthier. "Mesh Denoising based on Normal Voting Tensor and Binary Optimization". In: *IEEE Transactions on Visualization and Computer Graphics* 99 (2017), pp. 2366–2379. ISSN: 1077-2626.

[YRP18] Sunil Kumar Yadav, Ulrich Reitebuch, and Konrad Polthier. "Robust and High Fidelity Mesh Denoising". In: *IEEE Transactions on Visualization and Computer Graphics* (2018). ISSN: 1077-2626.

[Zhe+17] Yinglong Zheng et al. "Guided point cloud denoising via sharp feature skeletons". In: *Vis. Comput.* 33.6-8 (June 2017), pp. 857–867. ISSN: 0178-2789.

[Zhe+18] Yinglong Zheng et al. "Rolling normal filtering for point clouds". In: *Computer Aided Geometric Design* 62 (2018), pp. 16–28.

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich alle Hilfsmittel und Hilfen angegeben und die vorliegende Arbeit auf dieser Grundlage selbstständig verfasst habe. Die Arbeit wurde nicht in einem früheren Promotionsverfahren eingereicht.

Berlin, 2019

Martin Skrodzki

Zusammenfassung

Die Arbeit beschäftigt sich mit drei Bereichen, die jeweils im Umfeld der Verarbeitung von Punktwolken verortet sind. Der erste Bereich betrifft Konzepte von Nachbarschaften sowie zugehörige Datenstrukturen. In der Forschung wurde bereits mehrfach auf die Wichtigkeit von qualitativ hochwertigen Nachbarschaftsrelationen auf Punktwolken hingewiesen. Die Autoren von [LP05] berichten beispielsweise, dass die Ergebnisse ihres anisotropen Glättungsverfahrens stark von den Nachbarschaften abhängen, die sie für die prozessierte Punktwolke berechnen. Während bereits mehrere kombinatorisch oder metrisch basierte Konzepte für Nachbarschaften auf Punktwolken vorgestellt wurden, berücksichtigt keines dieser Konzepte die Informationen der Normalen, bzw. der Krümmung. Die Doktorarbeit stellt daher ein neues Verfahren für die Bestimmung von Nachbarschaften vor, die sich an der lokalen Form der Geometrie orientieren. Jegliche Nachbarschaftskonzepte sind nur dann von praktischer Relevanz, wenn sie auch schnell in unterschiedlichen Anwendungen berechnet werden können. Hierfür fällt die Wahl häufig auf die Datenstruktur der k-d Bäume von Friedman, Bentley und Finkel, s. [FBF77]. Dies liegt vor allem an der bewiesenen Laufzeit einer Nachbarschaftsabfrage von erwartet $\mathcal{O}(\log(n))$ auf n Punkten. Die Doktorarbeit präsentiert eine ausführliche Ausarbeitung dieser Laufzeitberechnung. Schließlich werden solche Datenstrukturen immer wichtiger, die von massiver Parallelisierung – z.B. auf Grafikkarten – profitieren. Eine solche stellt das Nachbarschaftsgitter dar, das von den Autoren von [Jos+09] eingeführt wurde. Die Doktorarbeit beantwortet mehrere offene Fragen zur Kombinatorik und zur Qualität der Nachbarschaften dieser Struktur.

Der zweite große Bereich der Arbeit zielt auf Mannigfaltigkeitsstrukturen für durch Punktwolken dargestellte Oberflächen. Immer, wenn die zugrundeliegende Geometrie die Struktur einer glatten Mannigfaltigkeit hat, kann erwartet werden, dass diese Struktur auch durch eine Stichprobe in Form einer Punktwolke abgebildet wird. Jedoch gibt es für Punktwolken bisher keine Konstruktion, die diese Struktur abbildet. Eine Lösung für diese Forschungslücke wird in der Doktorarbeit präsentiert. Außerdem ist das wichtigste Element für die Repräsentation von Mannigfaltigkeiten ein Atlas aus Karten mit entsprechenden Kartenwechselabbildungen. Die Doktorarbeit präsentiert einen Ansatz, bei dem Karten aus möglichst großen flachen Stücken der Punktwolke generiert werden. Diese flachen Stücke können dann einfach in anderen Verfahren genutzt werden.

Der abschließende dritte Teil der Arbeit beschäftigt sich mit effizienten und robusten Algorithmen für die Verarbeitung von Punktwolken. Wie oben bereits vermerkt, bieten Punktwolken – im Gegensatz zu Netzen mit ihren Flächenstücken – keine natürlichen Gewichtungen für ihre Elemente. Hier können nur die paarweisen Distanzen zwischen Punkten genutzt werden. Dies erschwert die Handhabung von Punktwolken mit uneinheitlicher Verteilung. Um diesem Problem entgegenzuwirken präsentiert die Arbeit neuartige Gewichte für die Verwendung in Diskretisierungen – z.B. von Differentialoperatoren – die für eine Verarbeitung wie im Falle einer einheitlichen Stichprobe sorgen. Abseits hiervon spielt in Anwendungen vor allem auch die Identifikation von Merkmalen einer Geometrie – Ecken, Kanten und Flächen – eine Rolle. Die Arbeit präsentiert daher einen auf dem Verfahren der bewegten kleinsten Quadrate basierenden Ansatz für die Erkennung von Geometriemerkmalen. Die Arbeit schließt mit der Vorstellung eines Glättungsalgorithmus für Punktwolken. Dieser entfernt bei der Akquise der Punkte fälschlicherweise erzeugtes Rauschen ohne dabei die oben angesprochenen Merkmale der Geometrie zu verwischen. Die Arbeit schlägt somit einen Bogen von der Erfassung von Punktwolken über theoretische Konstruktionen hin zur praktischen Verarbeitung.