Feature Aware Sampling and Reconstruction

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Xiaoyin Ge, M.S.

Graduate Program in Computer Science and Engineering

The Ohio State University

2015

Dissertation Committee:

Dr. Yusu Wang, Advisor

Dr. Huamin Wang

Dr. Rephael Wenger

© Copyright by

Xiaoyin Ge

2015

Abstract

Feature-aware sampling and reconstruction are important and widely studied topics in computer graphics. There has been a large amount of work proposed in the past decades for different applications. Nevertheless, the problems remain challenging. Among them, two of the challenges caught particular attentions. One is the lack of a universal definition of "feature" and the other is the lack of effective algorithmic tools to describe, extract as well as manipulate shapes and their features, especially for high dimensional complex data.

This dissertation work is dedicated to tackling these two challenges. First, we focus on two and three dimensional shapes, and propose a generic feature aware sampling strategy that is applicable in a general setting. Secondly, for more complex and potentially high dimensional shape/spaces, we provide a study focusing on reconstructing a specific family of feature structure – 1D skeleton, which has many applications in modern data analysis. Specifically, we achieve both of them by using geometric and topological methods.

Sampling a domain embedded in two or three-dimensional space is a fundamental topic in computer graphics. Recently, sampling methods with both randomness and uniformity have attracted great attention. At this frontier, blue noise sampling has been widely used for its effectiveness in reducing both the reconstruction artifacts and the information bias. However, most traditional uniform blue noise sampling

methods treat the sampling domain equally everywhere. Therefore, with a limited sample budget, they tend to put unnecessarily dense samples in the non-feature area while undersampling regions of features. We propose a general feature-aware blue noise sampling framework by considering a new metric which consists of both Euclidean distance and feature measurement. Unlike earlier feature-aware blue-noise sampling work that are only applicable to specific problems, our framework can be easily adapted to a range of applications with different notions of "features". We further demonstrate its effectiveness with three applications.

Given a sample set from a hidden domain, accurately recovering the domain or inferring features/structures behind it is a fundamental topic in modern data analysis. We are particularly interested in a specific family of features – 1D non-linear skeleton structure behind the data that occurs naturally in many application such as image silhouette identification, road network reconstruction from GPS data, modeling filamentary structure behind galaxies, as well as high-dimensional samples for the use of describing evolution of topics in documents or in tweeter database. Automatic extraction of such 1D-graph like features is not a trivial problem. In particular, it is usually hard to reliably identify junctions in the graph structure, especially for high dimensional data, and/or when noise are present. To this end, we propose to use a topological structure, called the *Reeb graph*, to help recover such graph-like features from potentially high-dimensional data usually coming in the form of unorganized point cloud. We further provide applications of our proposed methods, including one application in reconstructing the so-called singular surface in $\mathbb{R}^3$.

Finally, to handle the noisy data, we introduce a persistence based Reeb graph simplification strategy to remove small spurious branches or loops and provide theoretical study of it. In particular, we show that our method guarantees that no major topological features in the original graph will be destroyed during such simplification processes.

This is dedicated to my family for their love and support.

Acknowledgments

First, I would like to thank my advisor, Dr. Yusu Wang, for her support and guidance throughout my PhD program. She taught me how to find interesting topics, and how to think of the solution in a mathematically sound way. I also want to thank Dr. Tamal Dey, Dr. Rephael Wenger and Dr. Huamin Wang for their continuous advice in the past years.

I thank my collaborates for their help and mentoring. Especially I would like to thank Dr. Li-Yi Wei and Dr. Jack Wang. They gave me great help in the projects that I participated in. I also thank Dr. Leonid Sigal, Dr. Iain Matthews, Dr. Maryann Simmons, Dr. Aleka Mcadams and Dr. Rajesh Sharma for their mentoring in my summer intern program in Disney Research, which is a truly wonderful and memorable experience.

My graduate school days would not have been the same without the help, accompanying and encouragement from my group mates and friends. I want to thank Issam safa, Oleksiy Busaryev, William Harvey, Chuanjiang Luo, Lei Wang, Andrew Slatton, Fengtao Fan, Arindam Bhattacharya, Suyi Wang, Dayu Shi, Alfred Rossi, and Dong Zhe.

Finally, I would like to thank my parents for their love and selfless support. I devote this document to them.

Vita

June 2006	B.S. Electrical Engineering, Tongji University, Shanghai, China
December 2008	M.S. Electrical and Computer Engineering, The Ohio State University, Columbus, OH, USA
May 2014	M.S. Computer Science, The Ohio State University, Columbus, OH, USA
September 2009-present	Graduate Teaching/Research Associate, The Ohio State University, Columbus, OH, USA

Publications

Research Publications

Ulrich Bauer, Xiaoyin Ge, Yusu Wang "Measuring Distance between Reeb Graphs". *ACM Symposium on Computational Geometry, 2014 (SoCG 2014, Kyoto)*

Jiating Chen, Xiaoyin Ge, Li-Yi Wei, Bin Wang, Yusu Wang, Huamin Wang, Yun Fei, Kang-Lai Qian, Jun-Hai Yong, Wenping Wang "Bilateral Blue Noise Sampling". *ACM Transactions on Graphics, Volume 32 Issue 6, November 2013 (SIGGRAPH Asia 2013, Hong Kong)*

Tamal K.Dey, Xiaoyin Ge, Qichao Que, Issam Safa, Lei Wang, Yusu Wang "Feature-Preserving Reconstruction of Singular Surfaces". *Eurographics Symposium on Geometry Processing, Vol 31, 2012, Number 5 (SGP 2012, Tallinn)*

Xiaoyin Ge, Issam Safa, Mikhail Belkin, Yusu Wang "Data Skeletonization via Reeb Graphs". *Advanced Neural Information Processing Systems, 2011, 837-845 (NIPS 2011, Granada)*

Fields of Study

Major Field: Computer Science and Engineering

Table of Contents

List of Figures

Figure **Page**

Chapter 1: Introduction

Feature-aware sampling and reconstruction are two widely studied topics in computer graphics. Throughout literature, there has been a large amount of work proposed in the past decades for different applications. Even though the topics are broadly studied, challenges still remain. This work focuses on tackling two of them. One is about the lack of universally applied "feature" aware sampling algorithm. The other is about the lack of effective algorithmic tools to extract as well as to manipulate shapes and their features, especially for high dimensional complex data.

The first challenge is frequently seen in the applications such as sampling for image resize, downsampling for a dense point cloud, and global illumination based rendering. Traditionally, regular sampling or evenly spaced stochastic sampling pattern was used for its simplicity in implementation. However, these methods tend to cause undesirable side-effects, such as, the artifacts in the reconstructed data, or the missing of feature information due to an inappropriate sample distribution. To reduce the artifacts, a sequence of work has been proposed aiming at generating sets of randomly distributed samples from an input domain. Among them, the sample distribution complied with the so-called *blue noise* property is widely used. For the other side effect — poorly sampled features, however, so far there is no commonly applied method which can tackle the problem in a general way. This is partly due to

the fact that the definition of "feature" is highly application or domain dependent. Thus, it's hard to extend most of the existing empirical methods to a broader usage. To this end, we propose a general feature-aware blue noise sampling framework for lower dimensional space applications (2D or 3D), by considering a new metric which consists of both Euclidean distance and feature measurement. This forms the first part of this dissertation work.

The second part of this work makes an step towards to address the second challenge: developing effective algorithm tools to extract as well as manipulate shapes and features, especially for high dimensional complex data. Analyzing and extracting hidden structures for complex and potentially high dimensional shape/spaces has been not only a fundamental problem in computer graphics, but also ubiquitous in a broad rang of applications in engineering and scientific fields. With the rapid generating of diverse data, extracting geometric structure is often a crucial first step towards interpreting the data at hand. One of the prevailing ideas is to provide descriptor that encodes useful information about the hidden feature structure from the observed data. We provide a study focusing on reconstructing a specific family of feature structure – 1D skeleton, which has many applications in modern data analysis.

In the remaining part of this chapter, we present a brief introduction of the topics covered in this dissertation work.

1.1 Sampling

Sampling is a fundamental topic in computer graphics whose presence can be found across a wide range of fields, including image processing, mesh reconstruction and motion capture, etc. In recent years, sampling methods with both randomness

and uniformity have attracted a great amount of interests. At this frontier, blue noise sampling has been widely used due to its effectiveness in reducing both the reconstruction artifacts and the information bias.

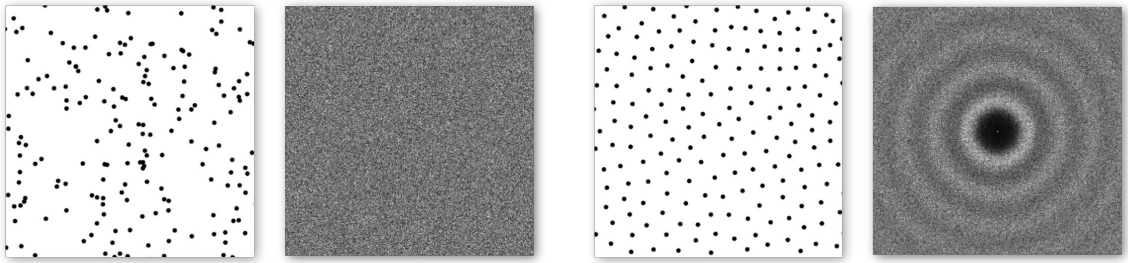
1.1.1 Blue noise

Before talking about the rational behind the first benefit of blue noise – effectiveness in reducing reconstruction artifacts (i.e., aliasing), we first give a brief review of the regular (grid-like) sampling pattern and the source of aliasing. Take 2D image sampling as an example. In a point-sampled image, the frequency band beyond the Nyquist limit (i.e., half of the sampling rate) are inadequately sampled. If the samples are regularly spaced, such high frequencies can appear as aliases. This usually occurs, for example, around the edges in the input image. To see how this happens, consider a one-dimensional sampling along the time axis t. Let a signal $f(t)$ be sampled at regular intervals of time, which is equivalent to multiplying the input signal with a periodic impulse train, a.k.a *shah* function, $\text{III}(t/T)$, where $\text{III}(t) = \sum_{n=-\infty}^{\infty} \delta(t-n)$ and δ is the Kronecker delta function. After that, information about the original signal $f(t)$ is preserved only at the sample points $f[n]$, where we use [.] to denote a discrete signal, and n is its discrete index. Its spectrum in frequency domain is a sequence of periodic replicas of the original signal. Based on the sampling theorem, if $f(t)$ contains no frequency above the Nyquist limit, then by passing an ideal lowpass filter (reconstruction filter), the sampled signal can be exactly recovered. However, for the signal with spectrum energy beyond the Nyquist band, its high frequency part can appear falsely as low frequencies due to the overlapping of the adjacent replicas, thus cause aliasing [63, 44].

The most straight-forward way to reduce aliasing is to increase the overall sampling rate. However, it seldom works in practices due to the fact that the frequency range of the input signal (usually with noise) is hard to predict. An alternative way is to change the sampling pattern (i.e., sample distribution). If the sample points are not regularly placed, the energy which causes aliasing in the reconstructed signal may reduce to be white noise – an artifact that is much less objectionable in humans visual perception system. An excellent example of non-regular sample distribution is human retina, which has a limited number of photoreceptor cells, but are not prone to aliasing. Those cells appear to have a random and locally uniform distribution that is usually referred as Poisson disk distribution [44], whose spectrum lacks the energy in lower frequency band (Figure 1.1(b), right) and is coincident with the concept of blue noise in terms of *The colors of noise* defined in *Telecommunications: Glossary of Telecommunication Terms*.

Due to such coincidence, in computer graphics, *blue noise* is usually used to refer a random sample distribution whose samples are "well separated". In other words, these samples are evenly distributed yet with stochastic properties. In practice, Poisson disk distribution can be generated by placing samples randomly with the restriction that no two samples are closer to each other than a certain distance. Its commonly seen implementations include dart throwing [81], energy based sample relaxation [28], or some variations like best-candidate sampling method [72], etc. We will give a further discussion about these methods in Chapters 2.

The second benefit of blue noise sampling is its effectiveness in reducing information bias compared with some other random pattern like white noise (Figure 1.1(a)). This can be inferred from the definition of Poisson disk sampling, where samples are



(a) (b)

Figure 1.1: (a) shows the sample distribution (left) and frequency domain spectrum (right) of white noise in 2D domain. (b) shows the corresponding distribution (left) and spectrum (right) of Poisson disk sampling pattern. All spectrum shown here are generated via 2D Fourier transform, and darker color denotes the area with less energy while lighter color refers the area with relatively higher energy. In the spectrum chart, the frequencies increases radiately from the center (0 frequency, a.k.a, DC component in signal processing context) towards the boundary.

well separated. Thus, with the same sample budget, the sampling with blue noise properties tends to give a better coverage of the entire domain.

1.1.2 Feature awareness

However, most traditional uniform blue noise sampling methods treat the sampling domain equally everywhere. Therefore, with a limited sample budget, they tend to put unnecessarily dense samples in the non-feature area while inadequately sample the feature region. The left figure in Figure 1.2 shows an example of the problems which can be potentially caused by using a uniform blue noise sampling in the image subsampling application. The thin features like the music lines are tend to be undersampled which leads to a problematic reconstruction result.

In the recent years, there have been several pieces of work dedicated to the topic of *adaptive noise blue sampling* [56, 14, 28, 24], namely, the sampling distribution

complies with a specific density distribution (not necessary to be uniform) — for example, the density defined based on the greyscale of an image or the curvature of a surface. One main drawback of these methods is that they are usually application dependent.

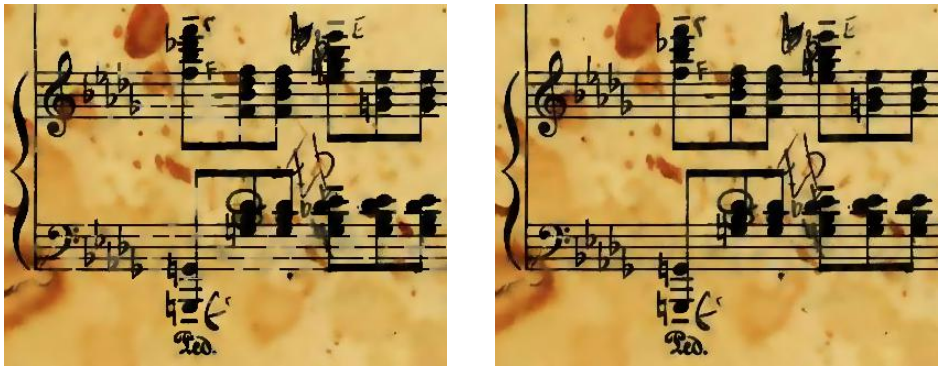


Figure 1.2: Left: image subsampling result by using uniform blue noise sampling. Some music lines are broken. Right: the corresponding result by using feature aware blue noise sampling. All lines and notes are well preserved.

Our contribution. We propose a general feature-aware blue noise sampling framework by considering a new metric which consists of both Euclidean distance and feature measurement. Unlike earlier feature-aware blue-noise sampling methods that are only applicable to specific problems, our framework can be easily adapted to a range of applications where "features" are differently defined. The right figure in Figure 1.2 shows the image subsampling result generated by using our feature aware blue noise sampling. As shown, comparing with the left figure (generated via traditional uniform blue noise sampling), our result gives a better reconstruction in terms of the image

features, such as the music lines and notes. We show the details of the algorithm and demonstrate its effectiveness through more applications in Chapter 2.

1.2 1D Feature Structure Recovering

Given a sample set from a hidden domain, accurately recovering the domain or inferring structures behind it is a long-standing topic in modern data analysis. At this frontier, geometric graph (1D skeleton structure) can serve as the underlying structure for modeling many natural phenomena from river/road networks, root systems for trees, to particle trajectories. For example, if we are interested in obtaining the road network of a city, we may sent out cars to explore various streets of the city, with each car recording its position using GPS devices. The resulting data is a set of potentially noisy points sampled from the roads in the city. Given these data, the goal is to automatically reconstruct the road network, which can be described as a geometric graph embedded in the two dimensional space.

The importance and applications of extracting hidden graph structure goes much beyond the GPS example we gave above. At a broad level, the graph-extraction problem is also related to manifold learning and nonlinear dimensionality reduction which has a rich literature, see e.g [9, 73, 77]. Manifold learning methods typically assume that the hidden domain has a manifold structure. An even more general scenario is that the hidden domain is a stratified space, which intuitively, can be thought of as a collection of manifolds (strata) glued together. Recently, there has been several approaches to learn stratified spaces [11, 45]. However, this problem is hard in general and requires algorithms both mathematically sophisticated and computationally intensive. To this end, graphs can be considered as a simplest complex structure

beyond manifolds — a one dimensional singular manifold structure. For this special family of singular manifolds, we develop an automatic and light-weighted algorithm to retrieve them from potentially high dimensional data.

More specifically, we proposed a Reeb graph based method to extract the hidden 1D feature structure from an unorganized input point set. We note that the concept of the *Reeb graph* has been used in a number of applications in graphics, visualization, and computer vision [13]. However, in the previous work, it has been typically used with mesh structures rather than a tool for analyzing unorganized point cloud data, especially in high dimensions, where constructing meshes is prohibitively expensive.

1.2.1 Reeb graph

Many problems in science and engineering can be posed in terms of real-valued functions, and the structure of such continuous function can be sometimes made explicit by considering the evolution of the components in the level set [39]. Reeb graph is one such topological concept.

Let $f : X \to \mathbb{R}$ be a continuous function defined on a topological space X as shown in Figure 1.3 (a). For each scalar value $\alpha \in \mathbb{R}$, the set $\{x \in X : f(x) = \alpha\}$ is called a *level set* of the function f. The Reeb graph of f is obtained by continuously collapsing every component of each level set into a single point. As α increases or decreases, those components appear, disappear, split and merge [47]. Reeb graph of f tracks such changes and provides a simple yet meaningful abstraction of the input scalar field. Several works have been proposed to implement fast Reeb graph computing in different applications [26, 70, 34, 47, 69].

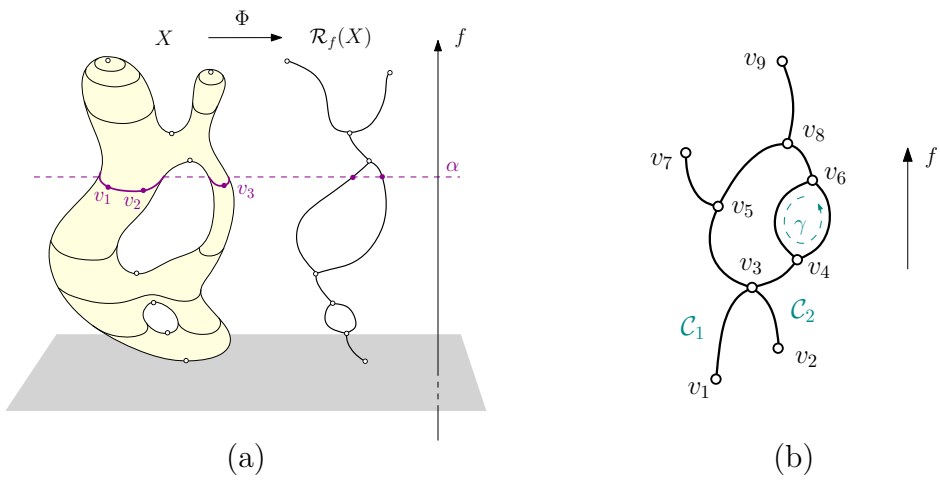


Figure 1.3: (a) For a topological space X with height function f as its scalar field, $\mathcal{R}_f(X)$ is its corresponding reeb graph. In the level set of $f(x) = a$, v_1 and v_2 belong to one component, and v_3 belongs to another. (b) Taking height function f as the scalar field, components $\mathcal{C}_1$ and $\mathcal{C}_2$ are generated when we sweep through minima v_1 and v_2, and they are merged when we arrive downfork saddle v_3, so that create *branching features*. The loop γ spanned by v_4 and v_6 is created when we pass through downfork saddle v_6 (*cycle features*).

Our contribution We proposed a light-weight Reeb graph based method to reconstruct the 1D skeleton hidden in the input (potentially high dimensional) point cloud data. Given a set of unorganized points, we start the process by first building its Vietoris-Rips complex (see Figure 1.4). The Rips complex intuitively provides an approximation of the underlying space that these input points are sampled from. It is popular in topological data analysis since its construction extends easily to higher dimensional space [86]. After constructing the complex, we then define a specific function on it, and retrieve its Reeb graph to approximate the hidden graph structure. This approach is very efficient, while at the same time it provides a reliable graph structure, especially in the junction area, which is typically hard for many prior approaches. In addition, a theoretical understanding of using Reeb graph to approximate metric graph has been provided in [21].

The algorithm we proposed has applications not only in low dimensional space, such as GPS road network reconstruction [42], 1d singular feature reconstruction in $\mathbb{R}^3$ [29], etc., but also be helpful in the analysis and description of potentially high-dimensional data [42]. See Chapter 3 and Chapter 4 for details.

1.2.2 Reeb graph simplification

Ideally, if the input point set is well organized (i.e., well approximate the hidden space), by choosing an appropriate scalar field f, the Reeb graph constructed in the above scenario catches the topology of the 1D skeleton structure. However, in practice, the point cloud usually comes with noise. To handle the noisy data, we introduce a persistence based Reeb graph simplification strategy to remove small "features" (i.e., branches and loops) potentially caused by the noise in the data [42].

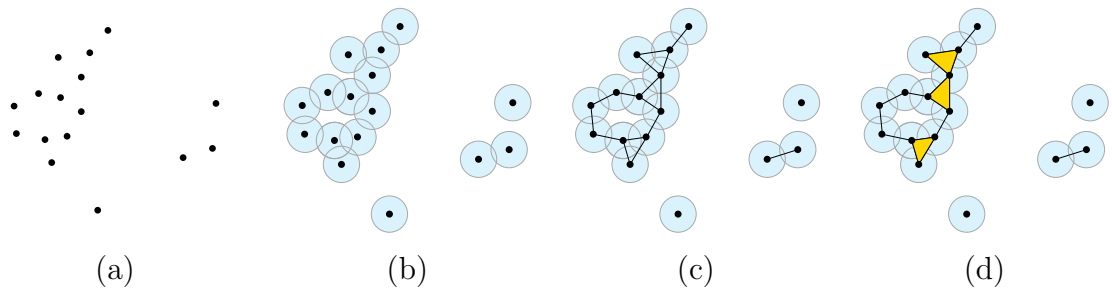


Figure 1.4: Above figure briefly shows the process of building the Rips complex of an input point set. Given a point set shown in (a), a radius r (shown as blue disk) is chosen and the neighbor relationship is built (b); (c)-(d) show the process of connecting the neighbor points and filling in the simplex (i.e., edge and triangle).

For Reeb graph $\mathcal{R}_f$, there is a natural way to define "features" and rank their "importance". That way turns out to be consistent with so-called persistent homology induced by the function $f : \mathcal{R}_f \to \mathbb{R}$. Since $\mathcal{R}_f$ is a graph, we only need to consider its 0- and 1-dimensional persistent homology, which corresponds to *branching feature* and *cycle feature*, respectively (See Figure 1.3(b)). The *importance* of these features can be measured according to their persistence, which is the absolute f difference between their birth and death events. More details would be provided in Chapter 3 and 5.

Our contribution. Noise in the input data can cause small spurious branching features and loop features. To remove them, we sort the graph features (both branch feature and cycle feature) in their increasing persistence order and set up a user-defined threshold δ. Intuitively, the branches and loops with persistence value less than δ will be categorized as "less important" and be merged (progressively) as shown in Figure 1.5.

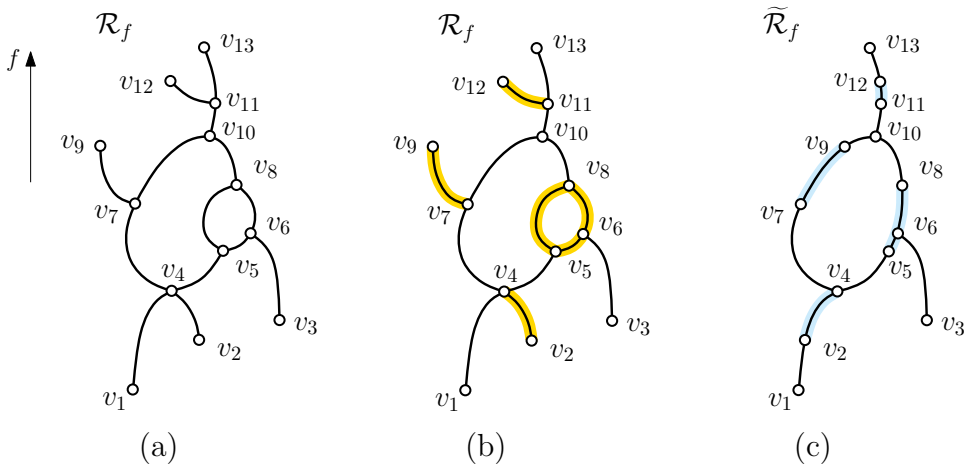


Figure 1.5: Reeb graph simplification. With the assumption that f is the height function, (a) shows a noisy Reeb graph $\mathcal{R}_f$. The spurious branches and loops with persistence value less than δ are highlighted in (b). (c) shows the simplified reeb graph $\widetilde{\mathcal{R}}_f$ with all these branches and loops removed.

We describe the details for the above simplification scheme for Reeb graph and demonstrate its effectiveness in Chapter 3 and 4. Furthermore, we present a theoretical study of the distortion caused by such simplification in Chapter 5. Let $\mathcal{R}_f$ and $\widetilde{\mathcal{R}}_f$ denote the original Reeb graph and the one after the simplification process, respectively, we will see that the distortion distance between $\mathcal{R}_f$ and $\widetilde{\mathcal{R}}_f$ would be smaller than $c\delta$, where c is a constant. In other words, it implies that the Reeb graph simplification approach we provide will not destroy major topological features in the original graph $\mathcal{R}_f$.

Chapter 2: Feature Aware Blue Noise Sampling

Blue noise sampling is a commonly used sampling method in graphics. As mentioned in Chapter 1, over the years, a variety of research efforts targeting both the characteristics and the generation of blue noise distributions have been conducted [27, 79, 81, 28]. In this chapter, we first review some usual blue noise sampling generating strategies. After that, we introduce a metric based blue noise sampling framework which can function as a generic feature aware stochastic sampling scheme. Part of the work presented in this chapter can also be found in the publication [43] and [22].

2.1 Blue Noise Sampling

In computer graphics, blue noise sampling generally refers to a sample distribution which is random yet uniform[1]. In the rest of this chapter we use "blue noise" to denote such distribution for abbreviation, though the original definition of blue noise may actually refer to a spectrum with less energy in low frequency band.

In prior work, the generating of blue noise can be roughly categorized into two families. One family uses a rejection-based scheme, called *dart throwing*. The other uses energy-driven approaches that can be further divided into a set of subcategories,

[1]Here, by "uniform", we mean an "evenly spaced distribution", instead of the distribution that has constant probability. The latter one is the concept usually used in probability theory.

such as Voronoi Diagram based ones [5, 59, 28, 84], kernel function based ones [67, 61], and so on.

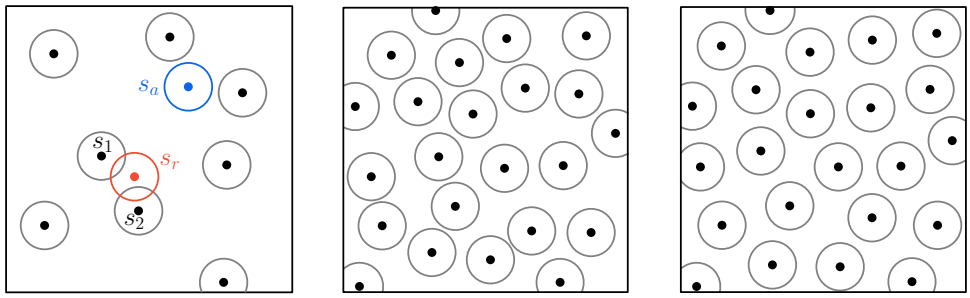
2.1.1 Dart throwing

Generally speaking, dart throwing algorithm aims to directly preserve the Poisson disk property: no two samples are closer to each other than a certain distance. The general idea of dart throwing is: Assume we have a random sample generator. Each time, it randomly generate a trial sample in the domain. If this trial sample's influence zone conflicts with the one of any sample which already exists in the domain, the trial sample will be rejected; otherwise, be accepted. Figure 2.1(a) shows this process. In the figure, the trial sample s_r will be rejected since its influence region (i.e., a disk centering at s_r) overlaps with the influence regions of existing sample s_1 and s_2, while the trial sample s_a will be accepted since no such conflict occurs.

The method is quite straightforward and, empirically, it produces a random as well as uniform sample distribution. However, it has some drawbacks. First, without an appropriate termination condition, the algorithm tends to be time consuming. This is due to the fact that the algorithm runs on a "rejection" based schema. Suppose in ideal case the domain can contain no more than N_{max} samples. When the number of accepted sample approaches N_{max}, the dart throwing process will suffer an obvious slowing down. That is caused by the fact that the trial sample could potentially have gone through a large amount of rejections before being finally accepted. To prevent it from happening, some termination conditions are commonly used, such as setting up a max trial sample count or a max failure count (the number of contiguous rejections), or both.

The second disadvantage is that it's hard to give a precise control of the final sample count. Because the sample generating process is a stochastic process, the number of samples that eventually get accepted can be case dependent. To see if the domain has contained reasonably dense samples, people empirically use a measurement $\rho = \frac{n}{N_{max}}$, where n is the actual sample count accepted, $N_{max} = \frac{D}{2\sqrt{3}r^2}$ is the maximum number of samples the domain can contain, D denotes the domain area, and r is the disk radius [54, 55]. The formula of N_{max} comes from the most compact sample pattern — hexagon tiling, where $2\sqrt{3}r^2$ is the area of each hexagon cell. A Poisson disk distribution with reasonably good quality usually has $\rho \in [0.65, 0.85]$. If ρ is too low, i.e., $\rho < 0.65$, the sample is over sparsely distributed. If ρ is too high, i.e., $\rho > 0.85$, a hexagon like regular pattern tends to appear, which conflicts with the randomness prerequisite. One thing worthy to note is that this ρ formula can also be rearranged as $\rho = \frac{r}{r_{max}}$, where r_{max} is the largest possible radius in case the actual sample count n is restricted, i.e., $r_{max} = \sqrt{\frac{1}{2\sqrt{3}n}}$. By setting ρ around 0.75, we can roughly compute an optimal r that empirically balances uniformity and randomness, therefore generates a reasonably good Poisson disk distribution. So, ρ, in some context, is also referred as "relative radius" [55]. Empirical though this schema is, it works reasonably well in practices. To some degree, it also eases the problem of uncontrollable final sample count.

Some other variant methods can handle "sample count" better, but the common trade-off is that the Poisson disk properties are less strictly met. Such methods include the soft disk dart throwing introduced in [81] and the best candidate algorithm in [72].



(a) (b) (c)

Figure 2.1: (a) shows the process of dart throwing algorithm. In each iteration, a trail sample is thrown out, if its neighbor area conflicts with the turf of the existing samples in the domain. The sample will be rejected, e.g., sample s_r will be rejected since its disk overlapped with the ones of the existed sample s_1 and s_2. s_a will be accepted since there is no such conflict occurred. (b) shows the result of dart throwing algorithm. (c) shows the relaxation result by taking (b) as the input.

Finally, the sample distribution generated via dart throwing may not have a satisfying local uniformity as shown in Figure 2.1(b). To this end, a *relaxation* process is usually implemented after the dart throwing process. In the relaxation, samples are slightly shifted around so that a better locally uniform distribution is achieved as shown in Figure 2.1(c). Some commonly used relaxation schemas, like Lloyd relaxation and kernel energy minimization, actually drop into the category of energy driven algorithm, which we will discuss soon.

According to the recently published survey work [85], the simplest data structure for uniform Poisson disk sampling is the quad-tree [83]. In this data structure, the cell size of the base grid is set to be $\sqrt{2}r$, so that each grid cell will receive no more than one sample. During the sampling process, the partially covered cells will be subdivided into smaller fragments in quad-tree. The follow-up work [37] further accelerated the sampling process by using a flat fragment array. Furthermore, GPU

based parallel computing strategies, e.g., [81, 14], are also widely used nowdays to speed up the dart throwing process.

2.1.2 Minimum energy driven algorithms

Minimum energy driven algorithms form another family of blue noise generating methods, though most of them are not initially aiming at generating a stochastic sampling distribution. In 2D case, the ultimate convergence state of these algorithms tends to be a regular sample pattern like hexagon tiling. The blue noise sample distribution is more like an intermediate state, which appears transitionally in the process of the convergence. We briefly introduce some commonly seen energy driven algorithms below.

Centroidal Voronoi tessellation Given a domain $\Omega \subseteq \mathbb{R}^N$, the set $\{V_i\}_{i=1}^k$ is called a tessellation of Ω if $V_i \cap V_j = \emptyset$ and the union of $\{V_i\}_{i=1}^k$ covers Ω completely. Given a set of points $\{\boldsymbol{p}_i\}_{i=1}^k$ in Ω, the Voronoi region (or sometimes referred as Voronoi cell) V_i corresponding to $\boldsymbol{p}_i$ is defined as [35]

$$V_i = \{\mathbf{x} \in \Omega \mid \|\mathbf{x} - \boldsymbol{p}_i\| < \|\mathbf{x} - \boldsymbol{p}_j\|, \forall j \neq i\} \ ,$$

where $\|.\|$ denotes the Euclidean distance in this context. Such tessellation and its dual, e.g., Delaunay triangulation, have numerous applications in computational geometry. The *mass centroid* $\mathbf{c}_i$ of region V_i is computed as

$$\mathbf{c}_i = \frac{\int_{V_i} \mathbf{x}\varrho(\mathbf{x})\, d\mathbf{x}}{\int_{V_i} \varrho(\mathbf{x})\, d\mathbf{x}} \ , \tag{2.1}$$

where $\varrho(\mathbf{x})$ denotes the domain density at point $\mathbf{x}$.

The discrete version of above definition is similar, but instead of having a continuous domain Ω, we now focus on the underlying domain which is represented as a set

of points, denoted by $\widehat{\Omega} = \{\mathbf{x}_i\}_{i=1}^m$ in $\mathbb{R}^N$. The Voronoi sets are now defined as

$$\widehat{V}_i = \{\mathbf{x} \in \widehat{\Omega} \mid \|\mathbf{x} - \boldsymbol{p}_i\| \leq \|\mathbf{x} - \boldsymbol{p}_j\|, \forall j \neq i, [\textit{tie-breaking condition}]\} \ ,$$

where [*tie-breaking condition*] is an optional condition, which works in case $\|\mathbf{x} - \boldsymbol{p}_i\| = \|\mathbf{x} - \boldsymbol{p}_j\|$. The corresponding mass centroid $\widehat{\mathbf{c}}_i$ is then defined as

$$\widehat{\mathbf{c}}_i = \frac{\sum_{\widehat{V}_i} \mathbf{x}\varrho(\mathbf{x})\, d\mathbf{x}}{\sum_{\widehat{V}_i} \varrho(\mathbf{x})\, d\mathbf{x}} \ . \tag{2.2}$$

Note, Eqn 2 works under the assumption that $\boldsymbol{P}$ is not restricted to be a subset of $\widehat{\Omega}$. Namely, $\boldsymbol{P} \subset \widehat{\Omega}$ is not necessarily true. Otherwise, the mass centroid could be more appropriately defined as [35]

$$\sum_{\mathbf{x} \in \widehat{V}_i} \varrho(\mathbf{x}) \|\mathbf{x} - \widehat{\mathbf{c}}_i\| = \inf_{\boldsymbol{p} \in \widehat{\Omega}} \sum_{\mathbf{x} \in \widehat{V}_i} \varrho(\mathbf{x}) \|\mathbf{x} - \boldsymbol{p}\| \ .$$

In practice, if the underlying discrete domain $\widehat{\Omega}$ is dense enough i.e., $|\widehat{\Omega}| \gg |\boldsymbol{P}|$, to simplify the computation process, one may first compute the "expected" mass centroid $\bar{\mathbf{c}}_i$, via Eqn 2.2, then find a $\boldsymbol{p}^* \in \widehat{\Omega}$ which is closest to this expected mass centroid as

$$\boldsymbol{p}^* = \arg\min_{\boldsymbol{p} \in \widehat{\Omega}} \|\boldsymbol{p} - \bar{\mathbf{c}}_i\| \ .$$

Then set $\widehat{\mathbf{c}}_i$ equal to $\boldsymbol{p}^*$.

There are several deterministic or stochastic approaches can be used to compute such tessellation, like Lloyd algorithm, sequential sampling algorithms, random sampling algorithm, and so on [35]. Here we give a brief introduction on Lloyd algorithm for its simplicity and wide usage. For other methods, one may refer to [35] for a thorough description.

Lloyd algorithm Lloyd algorithm, also sometimes called Lloyd relaxation in the literature, is an algorithm named after Stuart P. Lloyd aiming to find evenly-spaced points in subsets of Euclidean spaces, as well as partitions of these subsets into well-shaped and roughly equal-sized convex cells [60]. Lloyd relaxation in a continuous domain is an implementation based on the Voronoi cell partition; and its discrete version bears a resemblance to the k-mean clustering process.

In general, the algorithm of Lloyd relaxation can be summarized as following: Given a set of points $\boldsymbol{P} = \{\boldsymbol{p}_1, \cdots, \boldsymbol{p}_N\}$, a.k.a, *sites* in some context, three steps are repeated until the process converges or some user defined criteria are met.

1. compute the Voronoi diagram of $\boldsymbol{P}$
2. integrate/sum each Voronoi cell; and compute the mass centroid $\mathbf{c}_i$ for the Voronoi cell of $\boldsymbol{p}_i$
3. update the position of $\boldsymbol{p}_i$ to be $\mathbf{c}_i$

Kernel-based method Kernel function based approaches are another type of algorithms aiming at generating uniform sample/point distributions. In those algorithms, a global energy E is defined usually in the form of kernel function $k(.)$ as follows:

$$E = \sum_{\boldsymbol{p}_i, \boldsymbol{p}_j \in \boldsymbol{P}, i \neq j} f\left(k(\boldsymbol{p}_i, \boldsymbol{p}_j)\right) \ .$$

This energy E is then progressively decreased so that a locally even-spaced sample distribution is achieved. Spectral method [67] and graph Laplacian method [61] are two typical kernel function based algorithms. In both of them, Gaussian kernel $k(\boldsymbol{p}_i, \boldsymbol{p}_j) = e^{-g(\boldsymbol{p}_i, \boldsymbol{p}_j)}$ is used, where the non-negative function $g(\boldsymbol{p}_i, \boldsymbol{p}_j)$ indicates the difference between $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$. Figure 2.2 shows an example result from the Gaussian weighted graph Laplacian approach proposed in [61], where the input is a point cloud

in white noise distribution; and after several iterations, the point cloud become more and more evenly spaced.

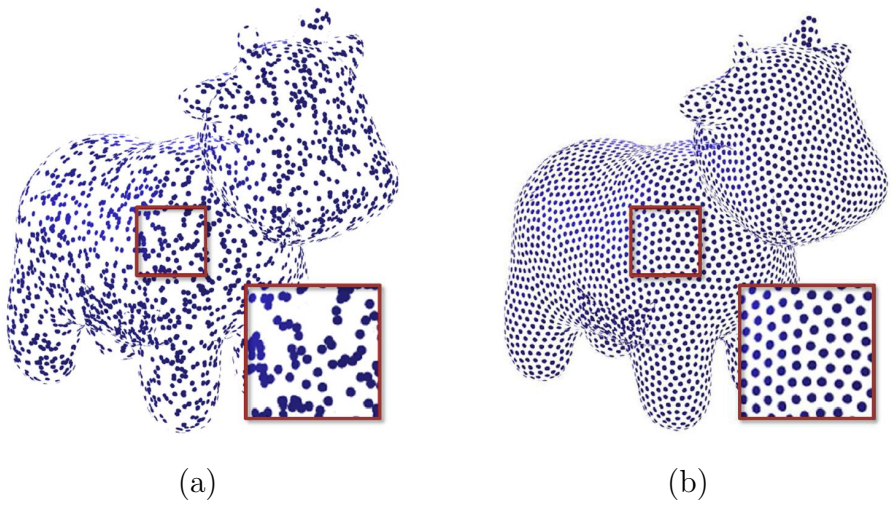


Figure 2.2: (a) shows the input point cloud $\boldsymbol{P}$ in white noise distribution. (b) shows the output of the approach proposed in [61], which relies on Gaussian-weighted graph Laplacian.

Kernel methods are prevalently used in surface remesh or 3D space point cloud uniformization, where the surface can either be explicitly given or be represented by a underlying point cloud [58, 67, 50, 61]. One major drawback of these energy minimizing strategies is that they tend to converge and terminate at local minimums instead of the global one. The direct consequence of such premature convergence is that the global evenness is hard to guarantee (in the final sample distribution), though the local uniformity is usually well achieved. To ease this problem, some auxiliary strategies are used, for example, adding a random shift to the sample position or a simulated annealing process in [61]. The later one moves points in a multi-scale

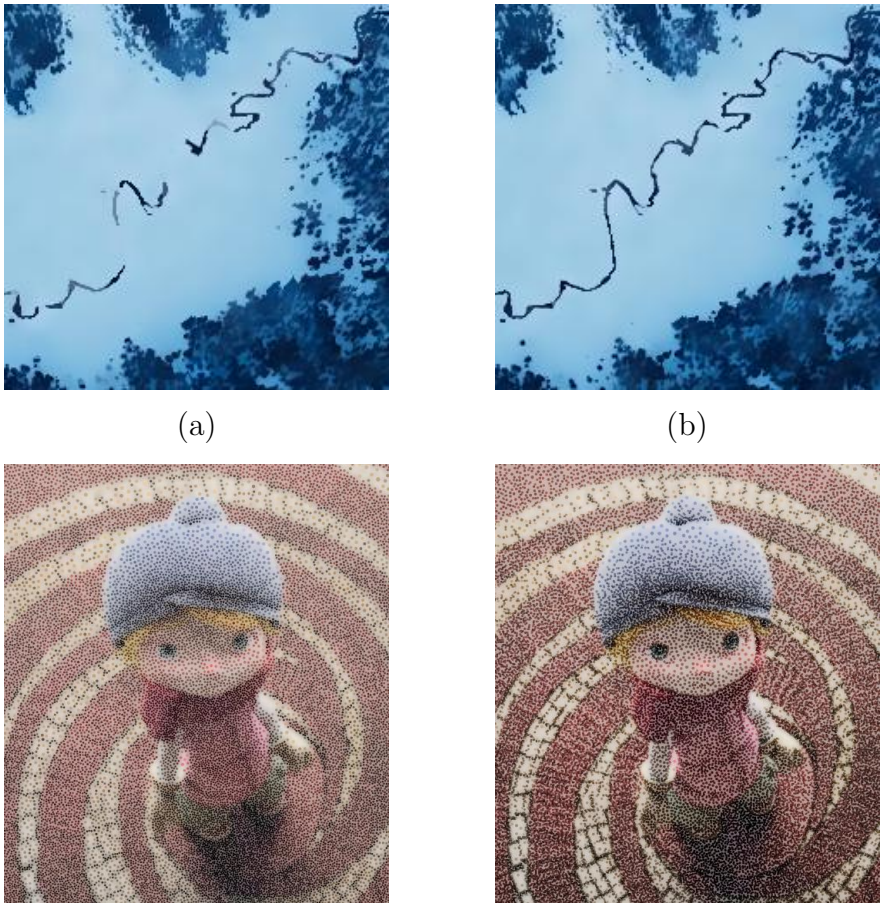
manner so that a better well-separateness properties can be obtained. An alternative way to solve this problem is, instead of taking an arbitrary distributed point set as the input, one can use the output of a dart throwing process, in which the global evenness has been well achieved.

2.2 Feature Aware Blue Noise Sampling

We have talked about some general methods used in blue noise sampling. The sample set produced by them covers the domain evenly, but it may potentially inadequately sample the feature area so that cause trouble in the subsequent processes, such as reconstruction. Figure 2.3(a) and (c) show the image subsampling and RGB image stippling result generated by using uniform blue noise sampling[2]. In Figure 2.3(a), to save the sample budget, the sample radius is set to be greater than the minimum river width. As a consequence, parts of the river are inadequately sampled and it leads to some noticeable errors in the reconstructed image. Similarly, without special care on the feature region, the girl's facial expression is blurred out in Figure 2.3(c).

Generally speaking, in the sampling process, given a fixed number of samples, the goal is to return a good coverage of sampling domain as well as a good representation of the *features* inside the domain. Here, the notion of "feature" is application varied. For example, in image stippling [5, 56, 40] or resizing [43], the *feature* is the difference between pixel color, and the goal is to distribute the stipples uniformly and yet depict the image boundary well. In 3D space point cloud downsampling application[67, 22], the *feature* can be the curvature of the underlying surface or the variation in surface

[2]The input image of the first row in Figure 2.3 and the input image of Figure 2.8 are from *National Geographic*. The input image of the second row in Figure 2.3 and the input image of Figure 2.6 are from a computer-animated short film *Alma*.



(a) (b) (c) (d)

Figure 2.3: Comparison between uniform blue noise sampling and feature aware blue noise sampling. (a) and (c) show the output of uniform blue noise sampling in image subsampling and RGB stippling application. (b) and (d) show the corresponding results generated via feature aware blue noise sampling.

normals, and the goal is to reduce the number of samples so that the subsequent reconstruction process could be less costly while the underlying surface are still be able to roughly accurately reconstructed.

There lacks a general method to combine these two components. To this end, we propose a general feature-aware blue noise sampling framework — *bilateral blue noise sampling*, mainly for lower dimensional space applications (i.e., 2D or 3D).

2.2.1 Bilateral metric

The core idea of our framework is a new metric which considers the spatial difference as well as the "differential" features between two samples $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$. Here we use "differential" feature to indicate a type of features which can be represented as the difference between the "property" value of two samples. The "property" here can be a scalar value or a vector as long as it's discriminative enough. For example, in the discrete case, surface curvature can be roughly illustrated as the normal difference between two neighbor surface points; and the boundary in an image can be identified via the color difference between two adjacent pixels.

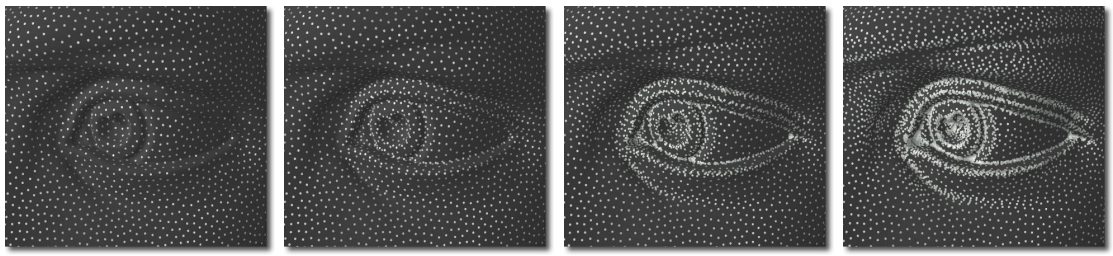
In particular, we define each sample $\boldsymbol{s}_i$ as a high dimensional vector $\boldsymbol{s}_i = [\boldsymbol{p}_i, \boldsymbol{\eta}_i]$, where $\boldsymbol{p}_i$ is $\boldsymbol{s}_i$'s position in Euclidean space, and $\boldsymbol{\eta}_i$ represents its property of interest. The bilateral metric (bilateral distance) is then in the form of

$$\xi(\boldsymbol{s}_i, \boldsymbol{s}_j) = f(g(\boldsymbol{p}_i, \boldsymbol{p}_j), h(\boldsymbol{\eta}_i, \boldsymbol{\eta}_j)), \tag{2.3}$$

where ξ is positively proportional to both the spatial function value $g(.,.)$ and the feature function value $h(.,.)$. Given this, one possible definition could be

$$\xi(\boldsymbol{s}_i, \boldsymbol{s}_j) = \left(\frac{1}{\sigma_p^2} \|\boldsymbol{p}_i - \boldsymbol{p}_j\|^2 + \frac{1}{\sigma_\eta^2} \|\boldsymbol{\eta}_i - \boldsymbol{\eta}_j\|^2 \right)^{\frac{1}{2}} , \tag{2.4}$$

where $\frac{1}{\sigma_p}$ and $\frac{1}{\sigma_\eta}$ work as the weights to balance the spatial and the feature terms. Eqn 2.4 shows that the bilateral distance ξ will increase if two samples either stay further away from each other, or become very different in terms of the feature value.



$\sigma_\eta \to \infty$ $\sigma_\eta = 25$ $\sigma_\eta = 10$ $\sigma_\eta = 5$

Figure 2.4: Above figure shows the bilateral blue noise sampling results by using various σ_η weight, where $\boldsymbol{\eta}$ is set to be surface normal.

An alternative but related definition of the bilateral metric could be

$$\xi(\boldsymbol{s}_i, \boldsymbol{s}_j) = \frac{1}{\sigma_p} \|\boldsymbol{p}_i - \boldsymbol{p}_j\|^2 \, e^{-\frac{\|\boldsymbol{\eta}_i - \boldsymbol{\eta}_j\|^2}{\sigma_\eta^2}} \quad . \tag{2.5}$$

In both Eqn 2.4 and Eqn 2.5, decreasing σ_η will emphasize the impact of the feature term. By setting σ_η approaching ∞, the bilateral distance ξ reduces to be the Euclidean distance between two samples. Figure 2.4 shows the sample distribution with various σ_η value, where a smaller σ_η implies that more samples will concentrate on the feature area, i.e., the higher curvature area in this example.

2.2.2 Bilateral blue noise sampling

The bilateral metric defined in Eqn 2.3 can be used in both dart throwing blue noise generating algorithm and the energy driven algorithms we introduced in Section 2.1.

Dart throwing. The bilateral dart throwing algorithm follows a similar process as the traditional dart throwing, but uses bilateral distance ξ instead of the original Euclidean distance for the conflict check. Note that since we only change the distance

ξ but not the conflict threshold r, our method can be orthogonally combined with not only uniform (r is a constant) but also adaptive radius setting ($r(\boldsymbol{s})$ depends on spatial sample position).

Lloyd relaxation. Being similar to the definition of CVT in Section 2.1.2, we write the CVT in bilateral distance case as

$$V_i = \{\mathbf{x} \in \Omega \mid \xi(\mathbf{x}, \boldsymbol{s}_i) < \xi(\mathbf{x}, \boldsymbol{s}_j), \forall j \neq i\} \ .$$

Note, since the bilateral metric is not isotropic, there is no guarantee that the Voronoi cell V_i would be in a convex shape. But in practice, we haven't been aware of any notifiable problem which could be potentially caused by the non-convex property. We compute the mass centroid in the same way as shown in Eqn 2.1.

Kernel function We define the global energy E as

$$E = \frac{1}{N} \sum_{i=1}^{N} e^{-\frac{\xi^2(\boldsymbol{s}_i, \mathbf{S})}{\tau^2}}$$

where $\mathbf{S}$ is the sample set $\mathbf{S} = \{\boldsymbol{s}_1, \cdots, \boldsymbol{s}_N\}$. By taking the dart throwing result as the input, the global energy E reduces quickly if an appropriate step size is selected.

2.2.3 Applications

In the rest part of this chapter, we will demonstrate our framework with three applications: stippling, image subsampling and surface reconstruction.

Stippling Stippling refers to a technique that uses small primitives (e.g. dots) to illustrate images [75, 5, 40]. The primitives can be of the same color (e.g.black) or from a small palette of colors. The later one works due to the fact that human visual

systems tend to blend multiple dots in local spatial regions. Such trick for trading off spatial for color resolutions has also been used in image halftoning where samples lie on discrete regular pixel grids [68, 16, 57].

For both stippling (continuous domain) and halftoning (discrete domain) applications, it has arrived a common ground that a sample set with blue noise distribution is more visually attractive. Furthermore, maintaining image structures or features is also highly desired.

Bilateral blue noise sampling can be applied for such feature-aware image stippling or halftoning by simply using the image color as features $\boldsymbol{\eta}$ in Eqn 2.4. Figure 2.5 and Figure 2.6 show our image stippling result in both greyscale and RGB case. As shown, our results have a better feature preservation comparing to the adaptive blue noise sampling results, while a better space uniformity than the halftoning work shown in [57].

Image subsampling Nonlinear filtering, such as bilateral and medial filtering, has a variety of applications. However, its computation process tends to be slower compared to linear filtering. To this end, various acceleration methods have been proposed for the acceleration (see e.g. [82, 23, 2, 41].

Among them, sub-sampling is a viable approach for filters. Banterle et al.[6] further demonstrated that the blue noise sampling offers the advantage of reducing aliasing. The method we proposed here offers a potential content-aware sub-sampling method for nonlinear filtering. In a nutshell, by treating image color as features $\boldsymbol{\eta}$ in Eqn 2.4, we perform bilateral blue noise sampling according to the underlying image content, and implement the filtering based on the samples we just obtained.

We propose two implementations of our method to meet different speed/memory expectations: the local one, in which sample pattern within a filter kernel is generated independently for each output pixel; and the global one, in which sample pattern of the entire output image is produced as a pre-process, from which samples locating within individual output kernels are used during the following filtration.

Quality speaking, bilateral sampling tends to produce better quality than uniform sampling, especially under sparse sample distributions. This is illustrated in Figure 2.7 and Figure 2.8 for perceptual image quality, and Figure 2.9 for numerical error measurements under a variety of images and parameter settings. From the experiments, it seems that the bilateral sampling can achieve similar quality to uniform sampling with fewer samples, i.e., usually 75/60% for local/global sampling.

Global sampling tends to be less noisy than local sampling, both visually and numerically. This is because in a global method, two adjacent output pixels can have overlaps in their filter tap/sample sets, providing extra coherence than the case where the filter sets are produced independently. We have observed that such sample set coherence may cause bias in a very sparse sampling setting (e.g. large missing chunks in Figure 2.7(e)), but in most cases, it outperforms a local sampling. For above experiments, more implementing details can be found in the technique report [43].

Surface reconstruction with various sample size Geometry sampling is another important component in computer graphics, and can be benefited by using a sample distribution that preserves both features and blue noise properties[67]. In particular, more samples are usually allocated around the features such as tips and

creases, so that a better reconstruction can be achieved. Furthermore, distributing samples with blue noise properties can avoid the potential biases. Our framework can be helpful here by simply using surface normal as the feature $\boldsymbol{\eta}$ in Eqn 2.4.

One application of our method is that it can preserve features reasonably well even though the sample budget is relatively tight. Figure 2.10 shows the surface reconstruction result for Bimba model, with sample size reducing from $95K$ to $7K$. As shown, even in the case with only $7K$ samples, the major features, like the bread and knot, are still roughly preserved.

In addition, our method is helpful in reconstructing thin features like the bowl model shown in Figure 2.11. Given a dense point cloud sampled on the surface of the bowl, the traditional uniform blue noise downsampling approach may have troubles in preserving the thin features such as the bowl wall, where samples in the inner and outer side of the wall are far from each other in terms of the bilateral distance but close enough to cause disk influence conflict if we only consider the Euclidean distance. This causes a poorly reconstructed surface as shown in the first row in Figure 2.11. However, using bilateral distance will ease the problem and give a better reconstruction in general[3].

[3]The bowl model is from *Autodesk 123D*.

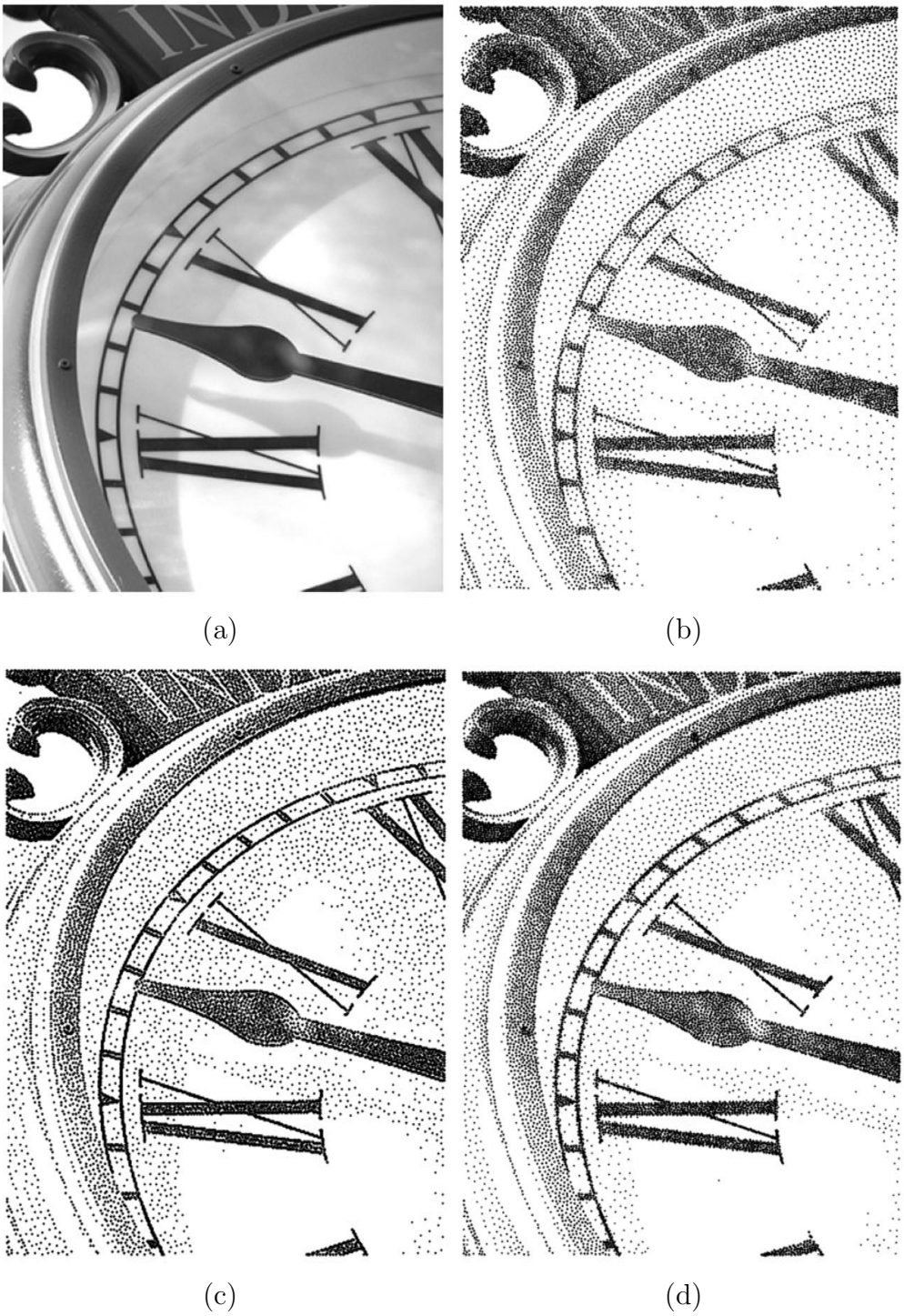


Figure 2.5: Stippling results. (a) shows the input image. (b) shows the result of adaptive Lloyd relaxation [81]. (c) shows the result of halftoning method in [57]. (d) is generated via bilateral Lloyd relaxation.

(a) (b) (c)

Figure 2.6: Stippling results. (a) shows the input image. (b) shows the result of adaptive Lloyd relaxation [81]. (d) is generated via bilateral Lloyd relaxation.

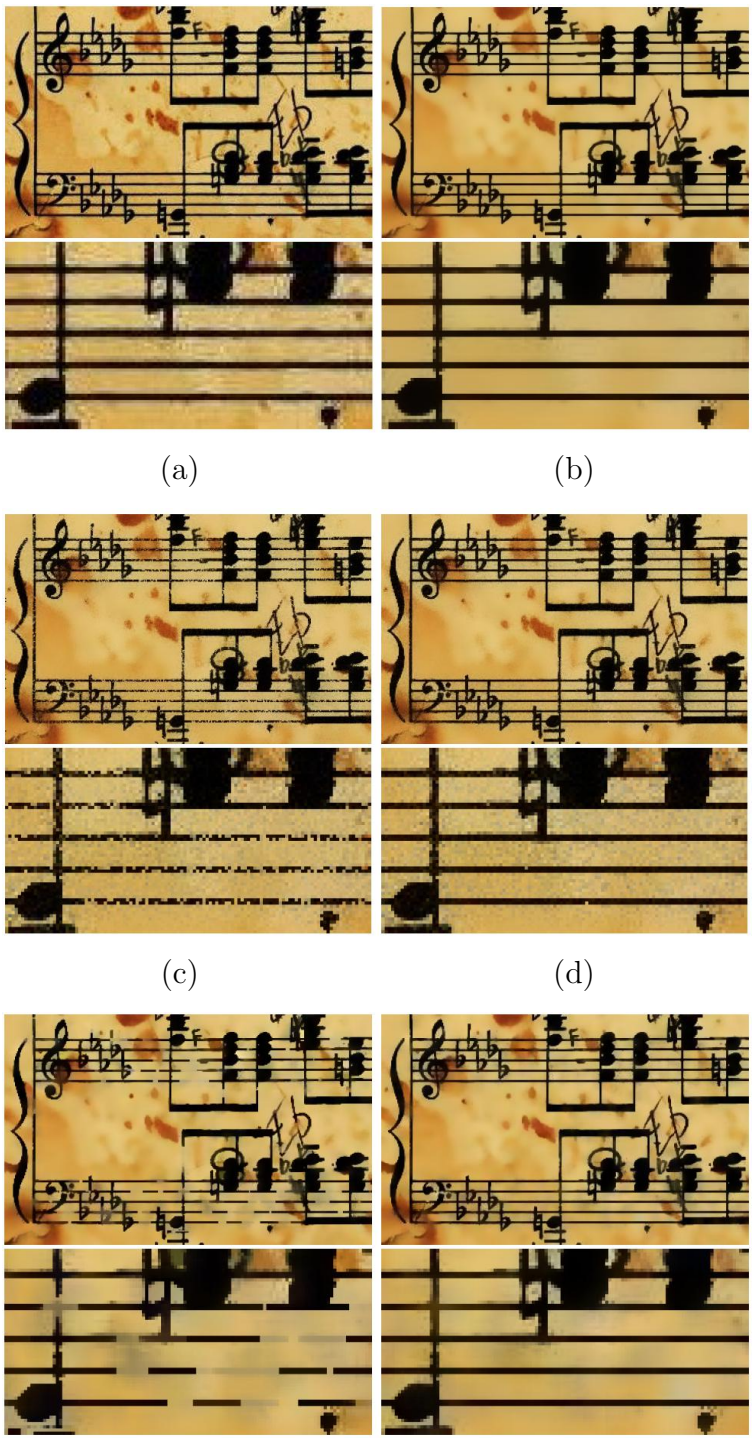


Figure 2.7: Image subsampling results — music sheet. (a) shows the input image. (b) shows the result of a full sampling. (c) shows the result of uniform sampling via local method. (d) shows the result of bilateral sampling via local method. (e) shows the result of uniform sampling via global method. (f) shows the result of bilateral sampling via global method.

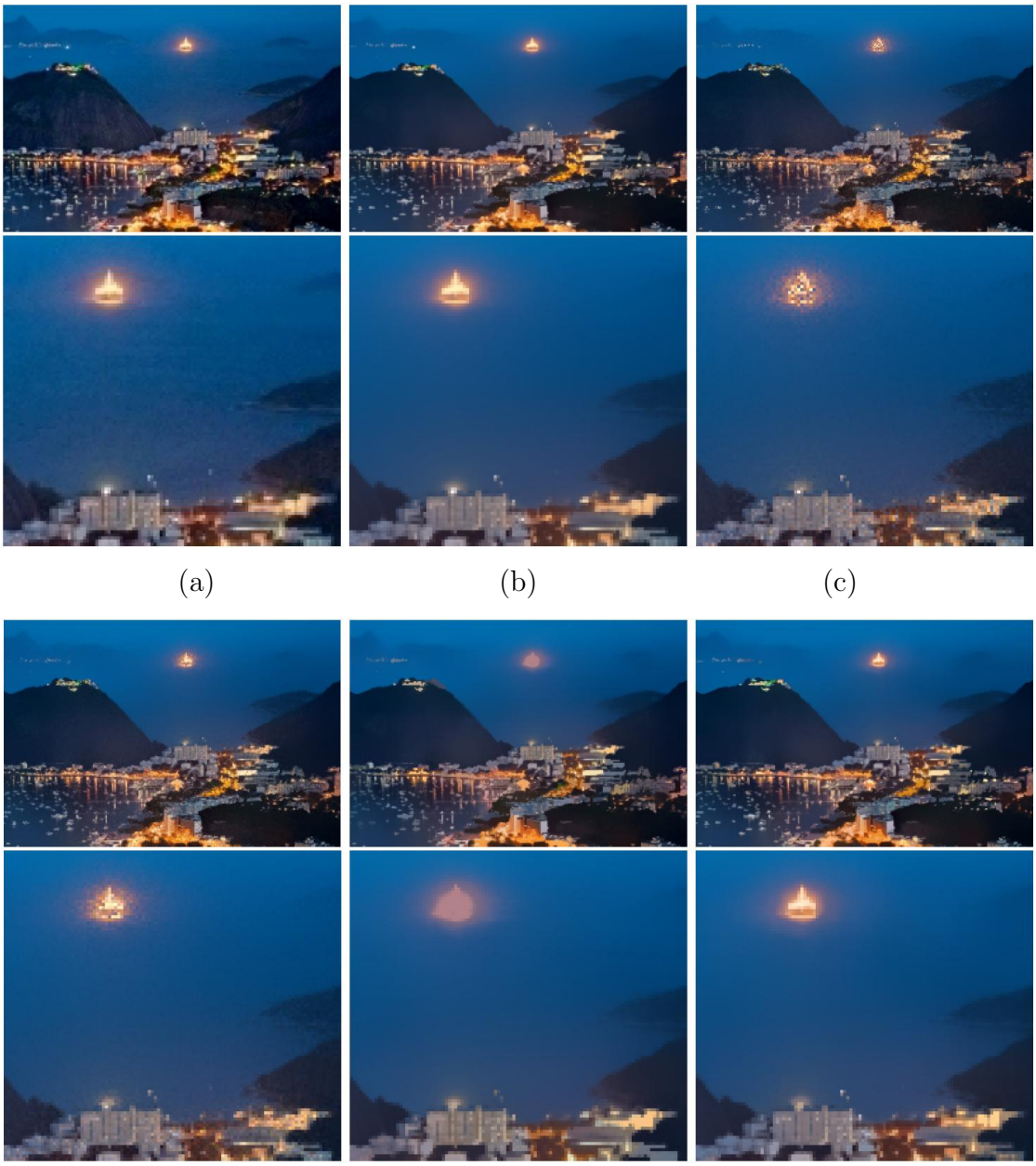


Figure 2.8: Image subsampling results — bay. (a) shows the input image. (b) shows the result of a full sampling. (c) shows the result of uniform sampling via local method. (d) shows the result of bilateral sampling via local method. (e) shows the result of uniform sampling via global method. (f) shows the result of bilateral sampling via global method.

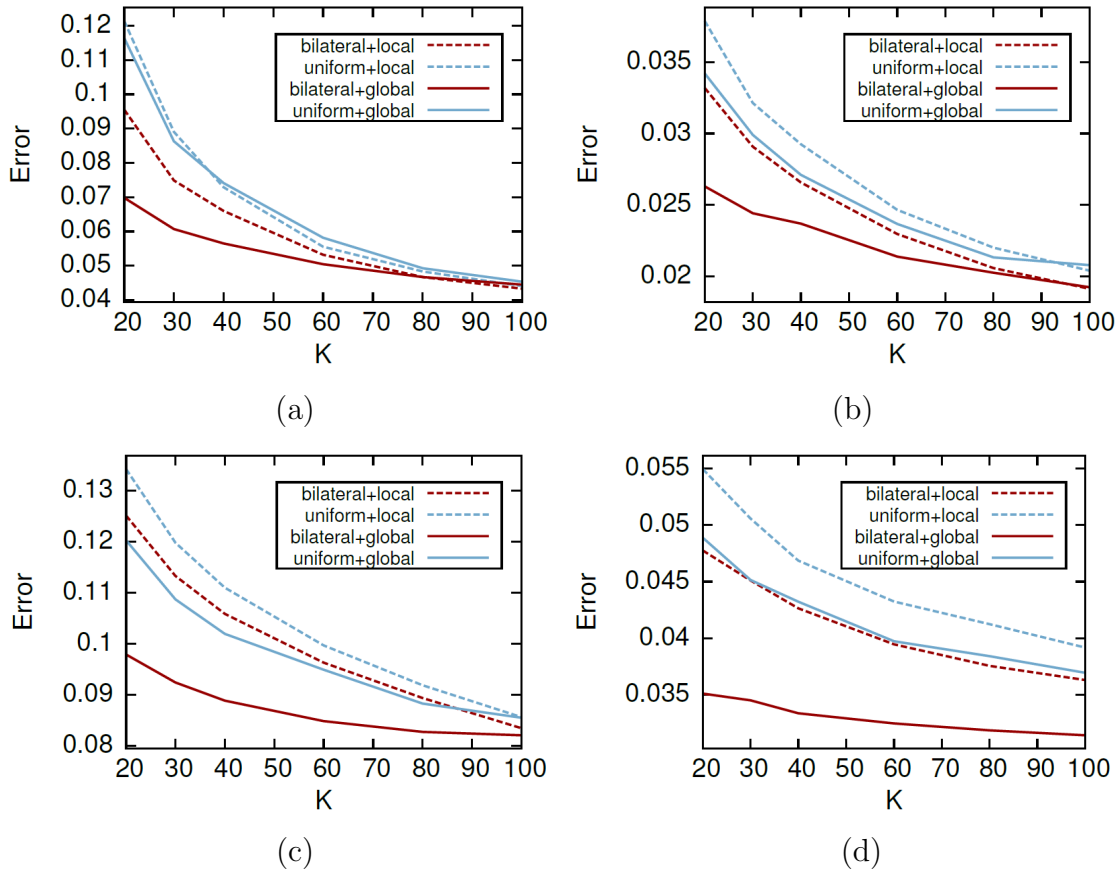


Figure 2.9: Image subsampling results quality comparison. (a) and (b) show the quality of "music sheet" result shown in Figure 2.7. (c) and (b) show the quality of the result of "bay" in Figure 2.8. The left column, i.e., (a) and (c), is generated by using per-kernel sample count $N_s = 0.5K$, where K denotes the kernel size (in the number of pixels). The right column, i.e., (b) and (d), is generated by using per-kernel sample count $N_s = 2K$.

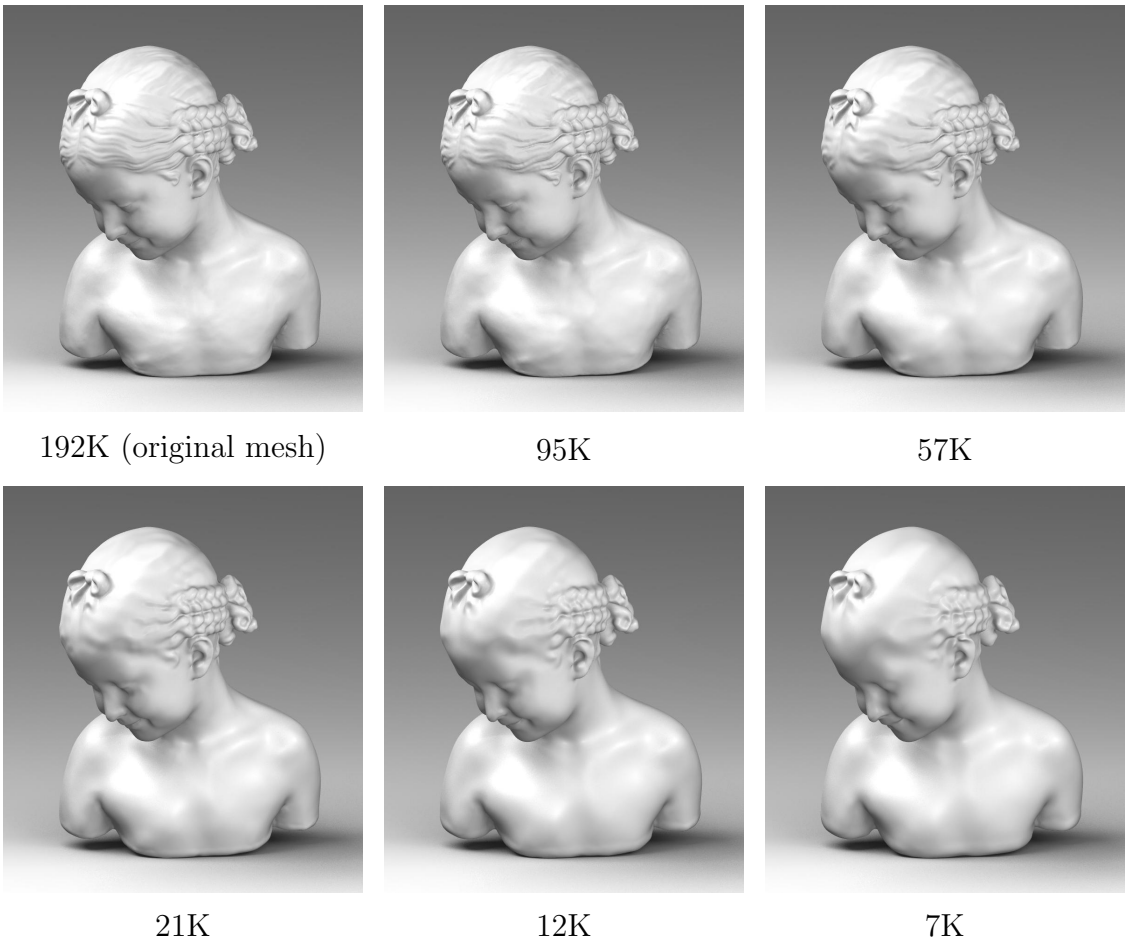
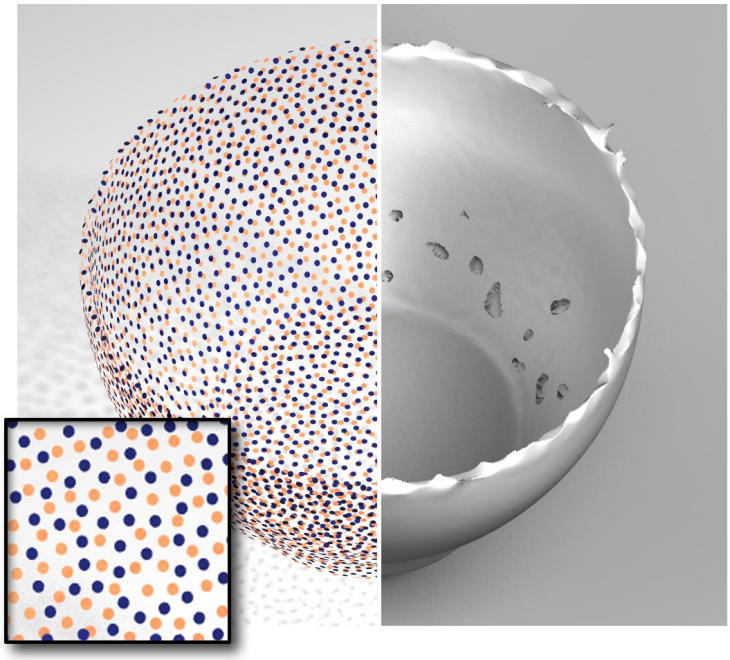
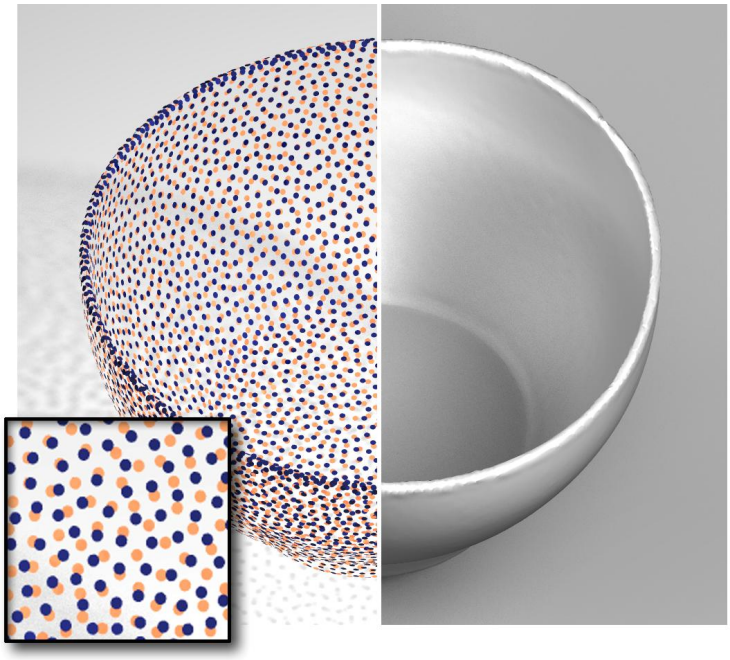


Figure 2.10: All results are produced by our bilateral dart throwing algorithm. As shown, our method can well preserve major features under a variety of sampling rates.



uniform blue noise sampling



bilateral blue noise sampling

Figure 2.11: The first row shows the sample distribution and final surface reconstruction result for the bowl model by using uniform blue noise sampling. The second row shows the corresponding results generated via bilateral blue noise sampling. In both images, blue dots and orange dots denote the samples on the inner and outer side of the bowl, respectively.

Chapter 3: 1D Feature Structure Detecting and Reconstruction

In last chapter, we introduced a general method for feature aware sampling in low dimensional space — 2D and 3D space. The method can be potentially extended to higher dimensional space by manipulating the feature term in the feature aware metric, i.e., bilateral metric ξ. Given such a sample set from a hidden domain, accurately recovering the domain (e.g., image or surface reconstruction) or inferring features/structures behind it (e.g., GPS roadmap extracting) has been a long-standing topic in graphics and many other data analysis related fields. For low dimensional applications, e.g., image or 2D manifold reconstruction, a variety of approaches have been proposed in the past decades. However, for more complex and potentially high-dimensional data, though attracts growing interest, this problem remains challenging.

In the complex and noisy non-linear data case, when the input data is high-dimensional, it is often of interest to approximate it with a low-dimensional or even one-dimensional space, since many important aspects of data are often intrinsically low-dimensional. Specifically, there are many scenarios where the underlying structure is graph-like, e.g, river/road networks or various trajectories. In this chapter, we introduce a framework to extract, as well as to simplify, a one-dimensional non-linear "skeleton" from unorganized data using a topological concept called the Reeb

graph. The algorithm we proposed is light-weighted, yet robust and effective. It does not require complex optimizations and can be easily applied to unorganized high-dimensional data such as point clouds or proximity graphs. In addition, it can also represent arbitrary graph structures in the data. We provide a number of experiments to demonstrate the effectiveness and generality of our algorithm, including the comparisons to the existing methods, such as principal curves. The work presented in this chapter can also be found in the publication [42].

3.1 Motivation

Geometric graphs are the underlying structures for modeling many natural phenomena from river/road networks, animal migration routes, root systems for trees, to blood vessels, and particle trajectories. An excellent example would be roadmap reconstruction from potentially noisy GPS data collected by commercial GPS devices. Nowadays, such data are quite popular and available at some open-streets project website like *http://www.openstreetmap.org*.

In addition, geometric graphs also arise from many modeling processes, such as molecular simulations. They can sometimes provide a natural platform to study a collection of time-series data, where each time-series corresponds to a trajectory in the feature space. These trajectories converge and diverge, which can be represented by a graph. Such graph in turn can then be used as a starting point for further processing (such as matching) or inference tasks.

Generally, there are a number of scenarios where we wish to extract a one-dimensional skeleton from an input space. The goal here is to develop, as well as

to demonstrate the use of a practical and general algorithm aiming at extracting graph structures from the input data (in any dimensions).

3.2 Related Work

At a broad level, the graph-extraction problem is related to manifold learning and nonlinear dimensionality reduction which has a rich literature, see e.g [9, 73, 74, 77]. Manifold learning methods typically assume that the hidden domain has a manifold structure. An even more general scenario is assuming that the hidden domain is a stratified space, which intuitively, can be thought of as a collection of manifolds (strata) glued together. This general problem is harder and requires algorithms both sophisticated mathematically and computationally intensive. By this token, we aim to learn a graph structure, which is simply a one-dimensional stratified space, allowing for simple approaches.

The most relevant previous work related to our graph-extraction problem is based an elegant concept of principal curves, originally proposed by Hastie and Stuetzle [48, 49]. Intuitively, principal curves are "self-consistent" curves that pass through the middle of the data. Since its debut introduction, there has been much follow-up work on analyzing and extending the concept and algorithms as well as on numerous applications. Please refer to e.g, [17, 36, 33, 53, 66, 76, 78, 80] among many others, for details. Below we discuss the approaches most relevant to the current work.

The original principal curves are simple smooth curves with no self-intersections. In [53], Kégl et al. represented principal curves as polygonal lines, and proposed a regularized version of principal curves. They gave a practical algorithm to compute such a polygonal principal curve. This algorithm was later extended into a principal

graph algorithm to compute the skeleton graph of hand-written digits and characters (see [52]).

Very recently in [66], Ozertem and Erdogmus proposed a new definition for the principal curve which associates it to the probability density function. Intuitively, one can imagine the probability density function as a terrain, then their principal curves resemble the mountain ridges. A rigorous definition can be made in terms of the Hessian of the probability density. Their approach has several nice properties, including the connections to the popular mean-shift clustering algorithm. It also allows for certain bifurcations and self-intersections. However, the output of the algorithm is only a collection of points lacking of the connectivity information. In addition, there is no explicit information showing which points should be the junction points (graph nodes) and which points belong to a single arc in the principal graph. Furthermore, the algorithm depends on reliable density estimation from input data, which is challenging for high dimensional data.

Aanijaneya et al.[1] recently proposed perhaps the first general algorithm to approximate a hidden metric graph from an input graph with theoretical guarantees. While the goal of [1] is to approximate a metric graph, their algorithm can also be used to skeletonize data. The algorithm relies on inspecting the local neighborhood of each point to first classify whether it should be a "branching point" or an "edge point". This approach has theoretical guarantees when the domain is nicely sampled and the parameters are correctly chosen. However, it is often hard to find suitable parameters in practice; in addition, the local decision, owing to its full reliance on the local information, tends to be less reliable when the input data contains bias (such

as a "fat" junction region). In Section 3.4, we show that our algorithm tends to be more robust in practical applications.

3.3 Reeb Graph Approach

Given a set of points $\boldsymbol{P}$ sampling a hidden domain X, we present a simple and practical algorithm to extract a skeleton graph G for X. In our case, the input points do not have to be embedded — we only need their distance matrix or simply a proximity graph as input to our algorithm.

The algorithm we proposed is based on using the *Reeb graph*, which provides a meaningful abstraction of the scalar field, and has been widely used in graphics, visualization, and computer vision. However, it has not yet been aimed as a tool to analyze high dimensional data from unorganized input data. By using the concept of Reeb graph in hidden structure extraction and data analysis, we can leverage the recent algorithms developed for computing and processing Reeb graphs, such as [30, 46, 69]. Moreover, combing the Reeb graph with the *Rips complex* allows us to obtain some theoretical guarantees for our approach (See Theorem 3.3.1 in Section 3.3.1).

3.3.1 Method

Given a topological space X and a scalar function f defined on this space, in Chapter 1.2.1, we have introduced the definition of the Reeb graph $\mathcal{R}_f(X)$ and Figure 1.3 (a) shows a simple example of Reeb graph with f as the height function. Alternatively speaking, $\mathcal{R}_f(X)$ can be defined as the image of a continuous subjective map $\varphi : X \to \mathcal{R}_f(X)$ where $\varphi(x) = \varphi(y)$ *if and only if* x and y come from the connected component in a level set of f.

The Reeb graph is an abstract graph whose nodes (i.e., critical points) indicate the changes in the connected components in level sets; and its arc represents the evolution of a connected component after it is generated but before it is merged, split or killed.

Computing Reeb graph in discrete setting. Assume the input domain is modeled by a simplicial complex K. Specifically, a *k-dimensional simplex* σ is simply the convex combination of $k+1$ independent vertices $\{v_0, \cdots, v_k\}$, and any simplex formed by a subset of its vertices is called a *face* of σ. A *simplicial complex* K is a collection of simplices with the property that if a simplex σ is in K, then any face of σ is also in K. A *piecewise-linear (PL) function* f defined on K is a function with values given at vertices of K and linearly interpolated within each simplex in K. Given a PL function f on K, its Reeb graph $\mathcal{R}_f(K)$ is decided by all the 0, 1, and 2-simplices from K (i.e., vertex, edge, and triangles from K). Hence from now on we use only up to 2-dimensional simplicial complex.

Given a PL function defined on a simplicial complex domain K, its Reeb graph can be computed efficiently via either a deterministic $O(mlogm)$ algorithm proposed in [69], or a randomized $O(mlogm)$ expected time algorithm proposed in [46], where m is the size of K. In the later one, the algorithm outputs the so-called *augmented Reeb graph* $\widehat{\mathcal{R}}_f(K)$, which contains the image of all vertices in K under the surjection map $\varphi : K \to \mathbb{R}$ introduced earlier. Figure 3.1 shows an example of such augmented graph $\widehat{\mathcal{R}}_f$ (right), its abstracted Reeb graph $\mathcal{R}_f$ (middle), and the simplicial complex domain K (left). In this case, the abstracted Reeb graph $\mathcal{R}_f$ only has four nodes, while the augmented Reeb graph $\widehat{\mathcal{R}}_f$ shows the image of all vertices $\{v_1, \cdots, v_{11}\}$.

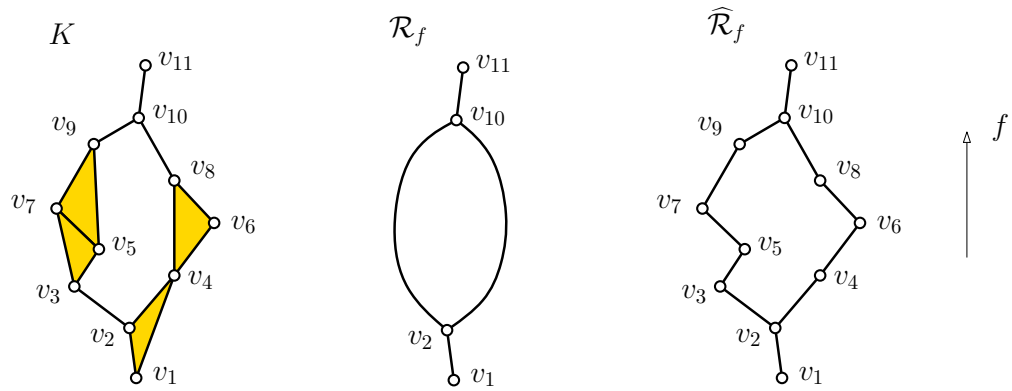


Figure 3.1: An example of the augmented graph $\widehat{\mathcal{R}}_f$ (right), its abstracted Reeb graph $\mathcal{R}_f$ (middle) and the simplicial complex domain K (left). To simplify the problem, we set the function f as the height field.

From the augmented Reeb graph, we can easily extract the junctions points (i.e., graph nodes in abstracted Reeb graph) and the regular points (i.e., vertices with degree 2) (e.g., vertices $\{v_2, v_3, v_5, v_7, v_9, v_{10}\}$ form the left arc between node v_2 and v_{10}).

Algorithm The basic algorithm we proposed here can be divided into two steps. First we set up the simplicial complex domain K. Then, we define a scalar function f on it, and retrieve its Reeb graph to approximate the hidden graph structure.

Step 1: Set up complex K***.*** The input data can be a set of points sampled from a hidden domain or a probabilistic distribution, a distance matrix, or, simply, the proximity graph among a set of points. In other words, the input points do not have to be embedded. Our goal is to compute (possibly an embedding of) a skeleton graph from the input data. First, we construct an appropriate space approximating the

hidden domain that input points are sampled from. We use a simplicial complex K to model such a space.

Specifically, given input sampled points $\boldsymbol{P}$ and the distance matrix of $\boldsymbol{P}$, we first construct a proximity graph based on either r-neighborhood or k-nearest neighbors (k-NN) information, where r and k are user-defined parameters; that is, a point $p \in \boldsymbol{P}$ is connected either to all its neighbors within $2r$ distance to p, or to its k nearest neighbors. We add all points in $\boldsymbol{P}$ and all edges from this proximity graph to the simplicial complex K. Next, for any three vertices $p_i, p_j, p_t \in \boldsymbol{P}$, if they are pairwise connected in the proximity graph, we insert the triangle $\triangle p_i p_j p_t$ into the complex K. Figure 1.4 shows the main steps of this building process (r-neighborhood). We remark that there is only one parameter involved in the basic algorithm, which is the parameter r (if we are using r-neighborhood) or k (if we are using k-NN) to specify the scale with which we look at the input data.

To further elaborate the motivation behind this construction, if the proximity graph is built based on r-neighborhood, then the above construction is simply the *Vietoris-Rips complex*, which has been widely used in manifold reconstruction (especially surface reconstruction) community to recover the hidden domain from its point samples. Intuitively, imagine that we grow a ball of radius r around each sample point. The union of these balls roughly captures the hidden domain at scale r.

The topological structure of the union of these balls is captured by the so-called *Čech complex*, which mathematically is the nerve of this union of balls. Hence the Čech complex captures the topology of the hidden domain when the sampling is appropriate (see e.g., [18, 65]). However, Čech complex is hard to compute, and the Vietoris-Rips complex is a practical approximation of the Čech complex that is much

easier to construct. Furthermore, it has been shown that the Reeb graph of a hidden manifold can be approximated with theoretical guarantees from the Rips complex [30].

Step 2: Reeb graph computation. Now we have a simplicial complex K that approximates the hidden domain. In order to extract the skeleton graph using the Reeb graph, we need to define a function g on K that respects its shape. It is also desirable that this function is intrinsic, given that input points may not be embedded.

To this end, we construct the function g as the geodesic distance in K to a certain base point $\boldsymbol{b} \in K$. We compute the base point by taking an arbitrary point $v \in K$ and choosing $\boldsymbol{b}$ as the point furtherest away from v. Intuitively, this base point is an extreme point. If the underlying domain indeed has a branching filamentary structure, then the geodesic distance to $\boldsymbol{b}$ tends to progress along each filament, and branch out at junction points. See Figure 3.2 for an example, where the thin curves are level sets of the geodesic distance function to the base point $\boldsymbol{b}$. Since the Reeb graph tracks the evolution of the connected components in the level sets, a branching (splitting in the level set) will happen when the level set passes through the saddle point v. In our algorithm, the geodesic distance function g in K is approximated by the shortest distance in the proximity graph (i.e, the set of edges in K). We then perform the algorithm from [46] to compute the Reeb graph of K with respect to g, and denote the resulting Reeb graph as $\mathcal{R}_g$. Recall that this algorithm in fact outputs the augmented Reeb graph

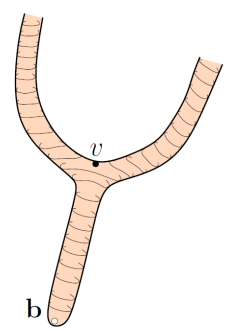


Figure 3.2: Geodesic distance function.

$\widehat{\mathcal{R}}_g$. Hence we not only obtain a graph structure, but also the set of input points (together with their connectivity) that are mapped to every graph arc in $\widehat{\mathcal{R}}_g$.

Theoretical guarantees. Given a domain X and a function $f : X \to \mathbb{R}$ defined on it, the topology of the Reeb graph $\mathcal{R}_f(X)$ may not reflect the topology of the given domain X. However, in our case, we have the following result which offers partial theoretical guarantee for aforementioned algorithm. Intuitively, the theorem states that if the hidden space is a graph G, and if our simplicial complex K approximates G both in terms of topology (as captured by homotopy equivalent) and metric (as captured by the ε-approximation), then the Reeb graph captures all loops in G. Below, we use $d_X(.,.)$ to denote the geodesic distance in domain X.

Theorem 3.3.1 *Suppose K is homotopy equivalent to a graph G, and $h : K \to G$ is the corresponding homotopy. Assume that the metric is ε-approximated under h; that is, $|d_K(x, y) - d_G(h(x), h(y))| \leq \varepsilon$ for any $x, y \in K$. Assume that $\varepsilon < l/4$, where l is the length of the shortest arc in G. Let $\mathcal{R}$ be the Reeb graph of K w.r.t the geodesic distance function to an arbitrary base point $\boldsymbol{b} \in K$. We have that there is a one-to-one correspondence between loops in $\mathcal{R}$ and loops in G.*

The proof of theorem 3.3.1 can be found in [42] Appendix A.

Time complexity. The time complexity of above algorithm is the summation of the time in computing (A) the proximity graph, (B) the complex K from the proximity graph, (C) the geodesic distance, and (D) the Reeb graph. (A) is $O(n^2)$ for high dimensional data (and can be made near-linear for data in very low dimensions) where n is the number of input points. (B) is $O(k^3 n)$ in case that each point takes k

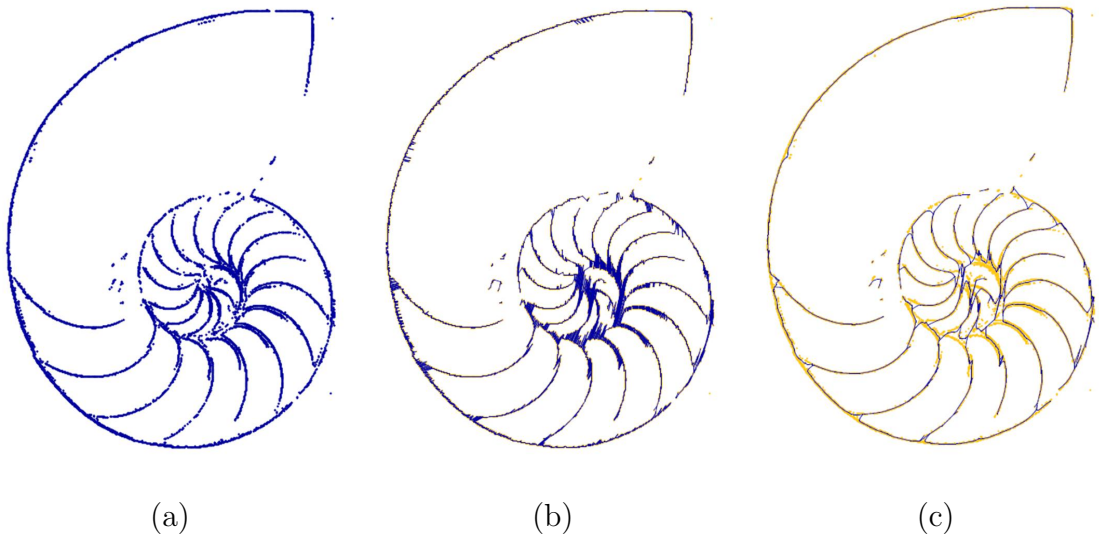


Figure 3.3: Overview of the algorithm. (a) is the detected edge points (blue) by implementing Robert edge filter. (b) shows the augmented Reeb graph (blue), where the edge points are in yellow. (c) shows the Reeb graph after a smooth/refinement process.

neighbors. (C) and (D) take time $O(m \log n) = O(k^3 nlogn)$ where m is the size of K. Hence, overall, the time complexity is $O(n^2 + k^3 nlogn)$. For high dimensional data sets, this time complexity is dominated by the computation of the proximity graph $O(n^2)$.

Embedding The Reeb graph is an abstract graph. To visualize the skeleton graph, we need to embed it in a reasonable way which reflects the geometry of the hidden domain. To this end, if points are not already embedded in 2D or 3D space, we project the input points $\boldsymbol{P}$ to $\mathbb{R}^3$ by using any standard dimensionality reduction algorithm. We then connect projected points based on their connectivity given in the augmented Reeb graph $\widehat{\mathcal{R}}$. Each arc of the Reeb graph is now embedded as a

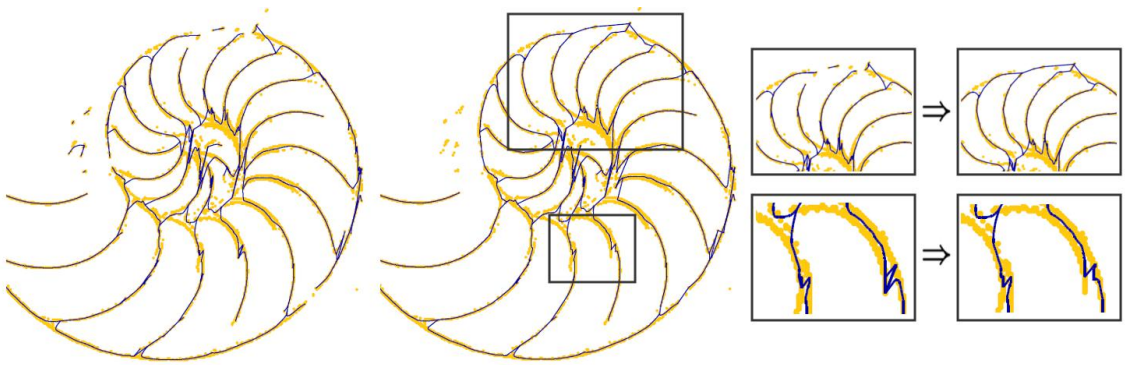


Figure 3.4: Left: the smoothed Reeb graph. Right: The fixing of the broken curves (upper) and the removing of the small noisy loops (lower) . The most right figure shows the zoomed in details.

polygonal curve. To further improve the quality of this curve, we fix its endpoints, and iteratively smooth it by repeatedly assigning a point's position to be the average of its neighbor's positions (Figure 3.3 (c))

3.3.2 Post processes

In practice, data can be noisy, and there may be spurious branches or loops in the Reeb graph $\mathcal{R}$ constructed. Following [3], there is a natural way to define "features" in a Reeb graph and measure their "importance". In this section, we present an intuitive interpretation of this method. A more detailed definition and description can be found in Chapter 5.

Graph simplification. Specifically, given a function $f : X \to \mathbb{R}$, imagine we plot its Reeb graph $\mathcal{R}_f(X)$ such that the height of each point $y \in \mathcal{R}_f(X)$ is the function value of all those points in X mapped to y. Now we sweep the Reeb graph bottom-up in increasing order w.r.t. the function value. As we sweep through a point y, we

inspect what happens to the part of Reeb graph that we already swept, denoted by $\mathcal{R}_f^y := \{w \in \mathcal{R}_f(X) \mid f(w) \leq f(y)\}$. When we sweep past a down-fork saddle s, there are two possibilities:

1. *Branching feature*: The two branches merged at s belong to different components C_1 and C_2. In such case, we say s ends a *branching feature*. The importance of this feature is the smaller height of the lower-branches being merged. See Figure 1.3 as an example, the down-fork saddle point v_3 merges the components C_1 and C_2, and generates a branching feature with the importance $|f(v_3) - f(v_2)|$.

2. *Cycle feature*: The two branches merged at s are already connected below s in $\mathcal{R}_f^s$. In such case, a new family of loops is created. This is called a *cycle feature*. Its size is measured as the smallest height of an loop formed with s in $\mathcal{R}_f^s$. Let γ denote such a loop. Its *height* is defined as $max_{z\in\gamma} f(z) - min_{z\in\gamma} f(z)$. See Figure 1.3, where v_6 is such a down-fork saddle, and the importance of loop γ is measured as $|f(v_6) - f(v_4)|$.

Now if we sweep $\mathcal{R}_f(X)$ top-down, we will obtain another set of branching-features and cycle-features captured by up-fork saddles in a symmetric manner. It turns out that these features (and their sizes) correspond to the so-called *extended persistence* of the Reeb graph $\mathcal{R}_f(X)$ with respect to function f [39]. The size/importance of each feature is called its *persistence*.

Detecting features and computing their persistence can be achieved in $O(nlog^2n)$ time, where n is the number of nodes and arcs in the Reeb graph [3]. We can now simplify the Reeb graph by merging features whose persistence value is smaller than

a given threshold δ (details see Chapter 5.3.1). This simplification process removes noise, and changes the topology structure of the Reeb graph. In Chapter 5, we show that the distortion caused by such modification can be bounded by $c\delta$, where c is a constant. In other words, the simplification process will remove the small loops and branches, but guarantees that no major topological features in the original graph will be destroyed.

Graph fixing. Finally, in case there is some missing data that causes missing links in the constructed skeleton graph, an extra fixing step is implemented. The fixing is achieved by connecting pairs of degree-1 nodes (x, y) in the Reeb graph whose distances $d(x, y)$ is smaller than a certain distance threshold. Here we use $d(x, y)$ to denote the input distance between x and y (if the input points are embedded, or the distance matrix is given), instead of the distance in the simplicial complex K which is constructed by our algorithm. Connecting x and y may either connect two disjoint components in the Reeb graph, thus creating new branch-features (Figure 3.4, right); or form new loop-features. We do not check the size of the new features created when connecting pairs of vertices. Small newly-created features will be removed in the subsequent simplification step.

3.4 Experiments and Results

In this section we first provide comparisons of our algorithm to some closely relevant prior methods. We then present three sets of experiments to demonstrate the effectiveness of our approach and show potential applications of skeleton graph extraction for data analysis.

(a) (b) (c)

Figure 3.5: (a) shows our result. (b) shows the result generated via PGA [52]. (c) shows the result generated via LDPC [66].

Methods to compare with. We compare our approach with three existing comparable algorithms (see Section 3.2 for more detailed description):

- the principal graph algorithm (PGA)[52];
- the local-density principal curve algorithm (LDPC) [66];
- the metric-graph reconstruction algorithm (MGR) [1].

Note that PGA only works for 2D images. LDPC only outputs point cloud at the center of the input data with no connectivity information.

In Figure 3.5, we show the skeleton graph of the image of a hand-written Chinese character. Our result is shown in (a). The output of PGA [52] and (the KDE version of) LDPC [66] are shown in (b) and (c), respectively. We see that the algorithm from [52], specifically designed for these 2-D applications, provides the best output.

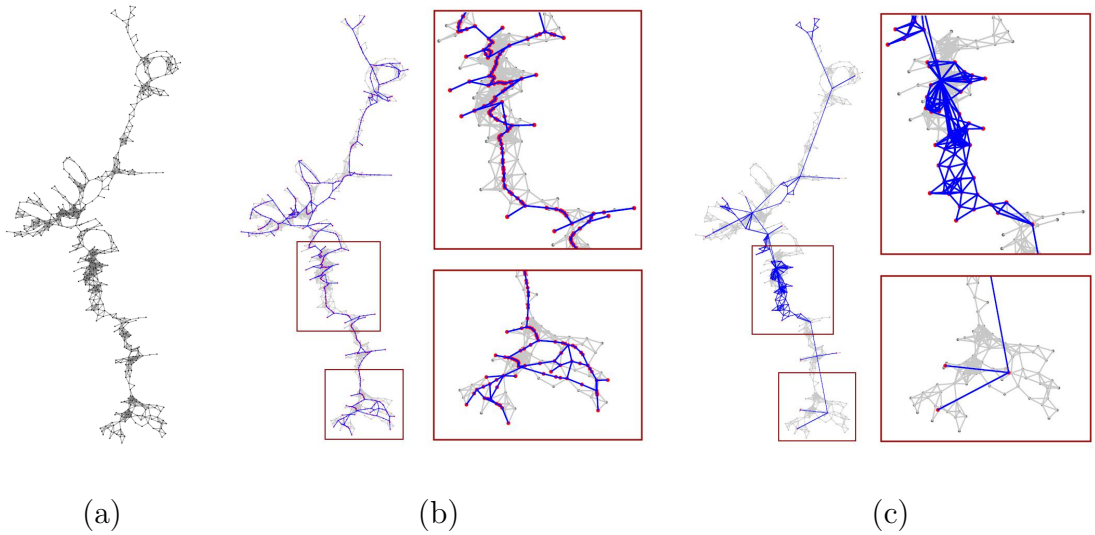
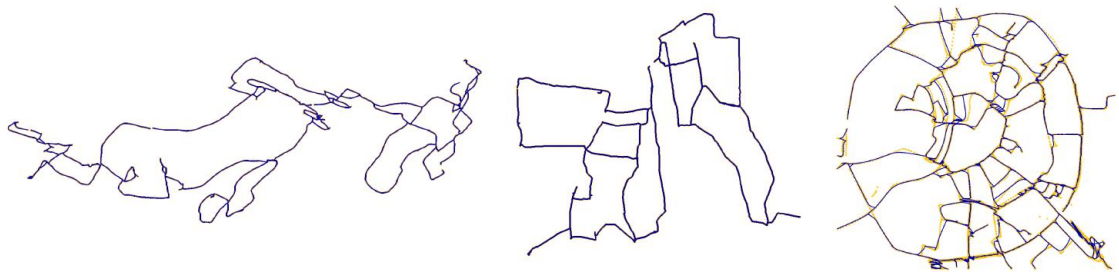


Figure 3.6: (a) shows the input image web. (b) shows our result. (c) shows the result generated via MGR [1]

However, the results of our algorithm, which is completely generic, are comparable. The output of LDPC is a point cloud (rather than a graph). In this example, many points do not belong to the 1D structure.

For the second set of comparisons we build a skeleton graph out of an input metric graph. Note that PGA and LDPC cannot handle such graph-type input, and the only comparable algorithm is MGR [1]. We use the image web data previously used in [1]. Figure 3.6 (b) is our output and Figure 3.6(c) is the output by MGR [1]. The input image web graph is shown in Figure 3.6(a), also shown in light (gray) color as the background of (b) and (c).

We further show two applications. One is for image edge detection (Figure 3.8 and Figure 3.3) and the other is for GPS road map reconstruction (Figure 3.7). The image edge samples (yellow points) are obtained by implementing a standard edge-detecting



(a) (b) (c)

Figure 3.7: (a) shows the reconstructed road map for *GPS trace 972358* from *openstreetmap.org*. (b) shows reconstructed map for *GPS trace 1004945*. (c) shows reconstructed result for the Moscow city (part).

algorithm, e.g., Robert edge filter or Sobel filter, onto the input image. The original GPS data in Figure 3.7 are denoted as yellow dots. In both cases (image edge and GPS trace), the final graphs are show in blue curve. In Figure 3.8, for completeness of the result, we keep all the graph components in the final result. Extra filtering processes can be used to easily remove the small components. In spite of these low dimensional applications, our method can also be useful in analyzing complex and potentially high-dimensional data. For example, it can be used to produce summary representation for multiple similar utterance trajectories (low and high-dimensional curves) for the purpose of trajectories alignment and discovering convergent and divergent portions of the trajectories [42]. In additions, in Chapter 4, we will show an extended application of our algorithm in singular surface reconstruction.

Combination with principle curve and LDPC algorithms. Finally, our algorithm can be used in combination with principal curve algorithms. In particular, one way to do this is to use our algorithm to first decompose the input data into different

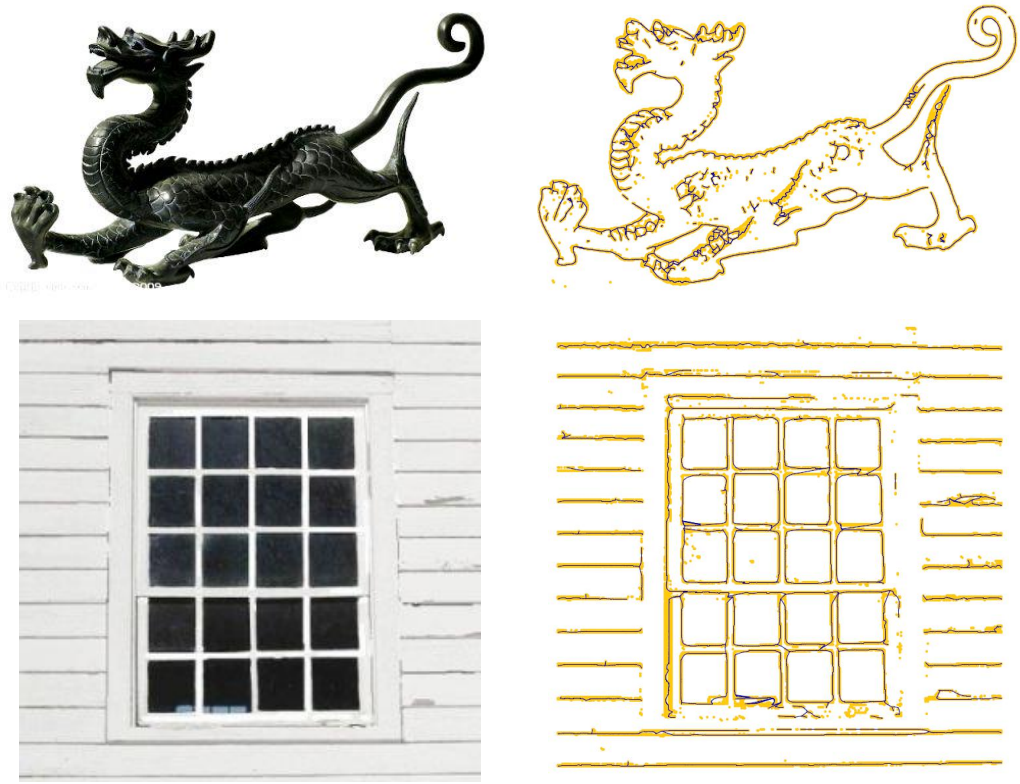


Figure 3.8: First row: the edge detection and reconstruction result for a dragon model picture. Bottom row: the corresponding result for a window picture. In both rows, the left column shows the input image and right column shows in the reconstruction edge graph.

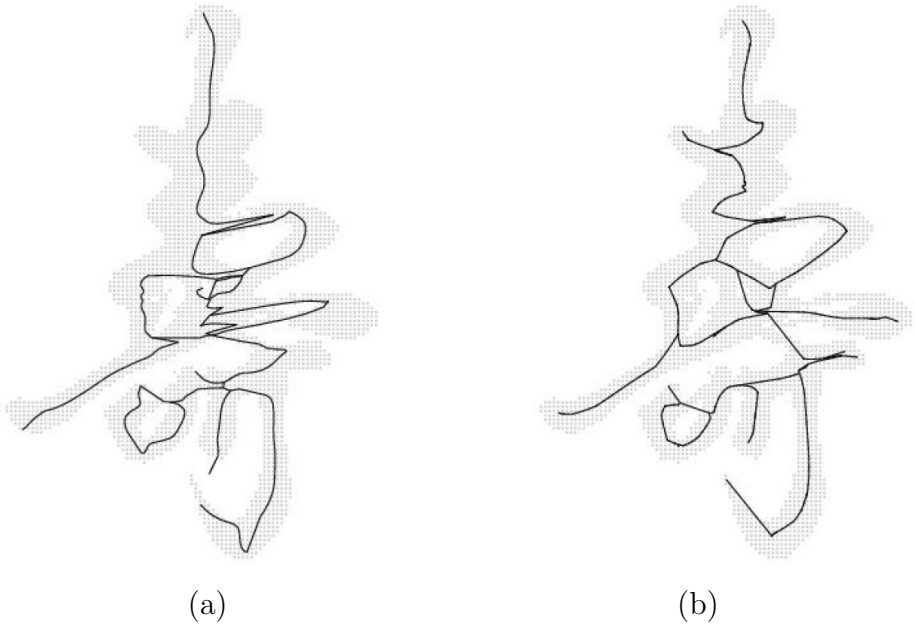


Figure 3.9: (a) shows the result generated by suing our algorithm to seed the principal curve algorithm in [53]. (b) shows the result generated by using LDPC algorithm [66] to seed our algorithm.

arcs of a graph structure, and then use a principal curve algorithm to compute the embedding of this arc locating in the center of the points contributing to it. An example is shown in Figure 3.9(a), where we apply the polyline principal curve algorithm from [53] to each branch output by our algorithm. We remark that we simply put these two algorithms together in a straightforward way, the result may not be ideal yet. For example, since we do not have any control on the endpoints of the branches when the principal curve algorithm from [53] is applied, the endpoints of the graph arcs may drift away and no longer meet. It will be an interesting future direction to see how to best combine our algorithm with a principal curve algorithm.

Alternatively, we can first use the LDPC algorithm [66] to move points to the center of the data, and then perform our algorithm to connect them into a graph structure. An example of such implementation is shown in Figure 3.9(b), where we apply our algorithm to the output of LDPC (using $\sigma = 2$ in KDE density estimation).

Chapter 4: Singular Surface Feature Reconstruction

In Chapter 3, we introduced an algorithm for 1D feature reconstruction. In this Chapter, we show how it helps in singular surface reconstruction which recently catches growing interest.

Reconstructing a surface mesh from a set of point samples is a long-standing topic in geometric modeling. It becomes challenging in presence of boundaries and sharp features. Furthermore, in practice, the point cloud data can be sampled from more general domains, including non-manifolds where multiple surface patches meet or intersect with each other. In the work of [29], also in the following chapter, we consider the so-called *singular surfaces* which consist of a collection of smooth surface patches with boundaries. These surface patches can intersect, or be "glued" along their common boundaries (i.e., sharp crease lines). For simplicity, we unify boundaries, intersections, and sharp creases under the aegis of singularities, and refer to them as feature curves. To date, an effective and practical algorithm to recover a singular surface from point data is still missing.

4.1 Our Approach

We propose a simple yet effective reconstruction algorithm for singular surfaces that can handle all three aforementioned singularities in a unified framework. Our

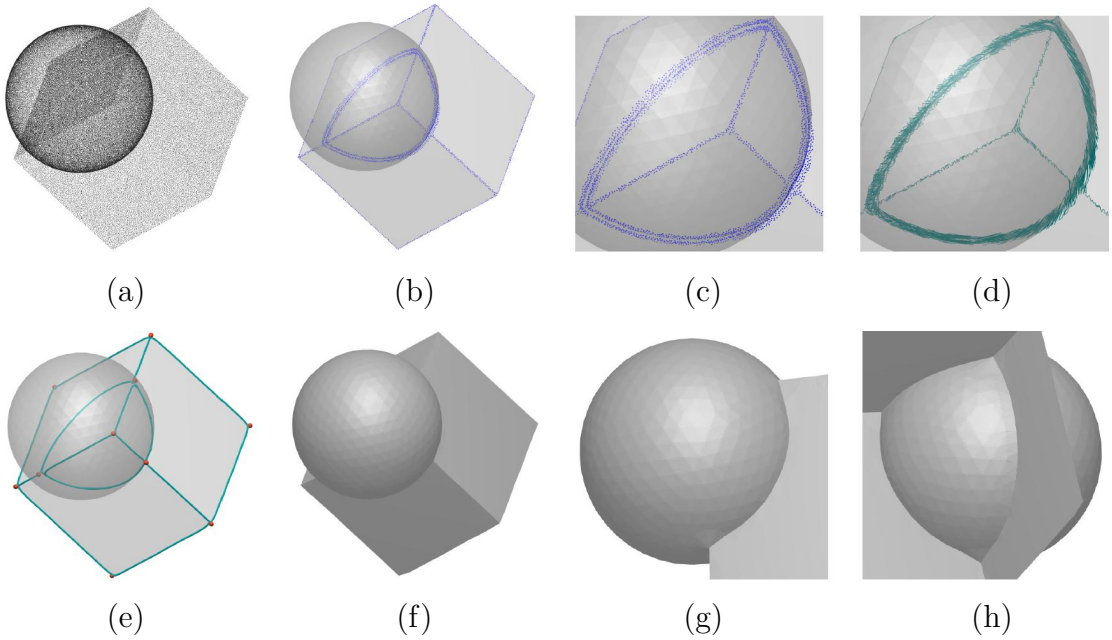


Figure 4.1: The workflow of the singular surface reconstruction framework. (a) Input point cloud. The hidden domain is a sphere intersecting a half-cube with only three faces. (b) Feature points are identified (in purple). (c) A zoom-in view of feature points around intersections. (d) Coarse feature curves are reconstructed. (e) Refined feature curves, with junction nodes and sharp corners marked as red points. (f) Reconstructed singular surface. Two zoomed-in views of reconstructed model near intersections in (g) and (h).

algorithm has two components (see Figure 4.1): (1) feature curve identification and reconstruction, and (2) singular surface reconstruction that respects these features.

For the first step, we employ a novel combination of the Gaussian-weighted graph Laplacian [10] and the Reeb graph we introduced in Chapter 3. This approach has several nice properties. First, it is general: it provides a unified approach to handle all three types of singularities. Second, it is simple: only the proximity graph from input points is required, and it does not involve estimating normals or tangent spaces, which could be unreliable around singularities. Furthermore, higher-order singularities, such as junction nodes where multiple feature curves meet, are automatically and reliably detected without any ad-hoc handling. Our approach is robust to noise and can possibly be used in other applications involving point cloud processing, such as in stylish drawing [71].

4.2 Singular Curves Identification and Reconstruction

4.2.1 Algorithm overview

Suppose we have a collection of smooth 2-manifolds with boundaries $\{\Omega_1, \cdots, \Omega_k\}$ isometrically embedded in $\mathbb{R}^3$. We call each Ω_i a surface patch. These surface patches may intersect each other in their interior, which gives rise to intersection feature curves. They can also be "glued" along (part of) their boundaries, producing the sharp-feature curves. Finally, we call boundary curves that are not shared by multiple manifolds as boundary-feature curves.

See Figure 4.2 and Figure 4.1 (e) for illustrations of different types of feature curves we consider. A regular point refers to a point not on these feature curves. Given a set of points $\boldsymbol{P}$ sampled from the singular surface $\Omega = \Omega_1 \cup \Omega_2 \cup \cdots \cup \Omega_k$,

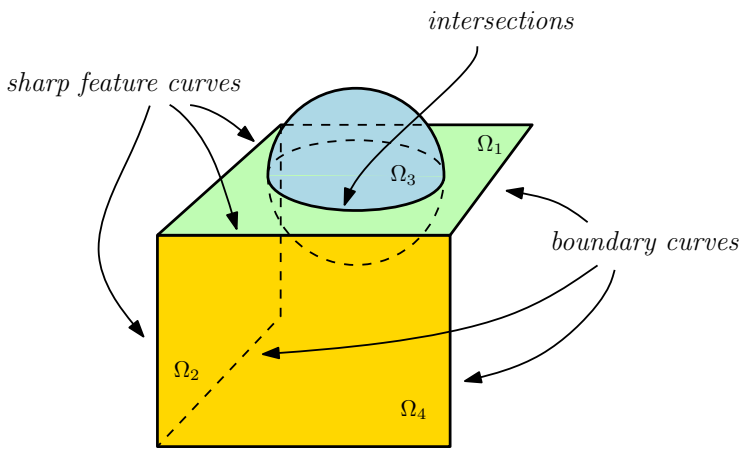


Figure 4.2: Above figure shows different types of singularity, including surface patches intersections, sharp feature curves, and boundary curves.

which is the union of Ω_is, our goal is to reconstruct Ω while preserving various types of feature curves. To this end, we use a combination of Gaussian-weighted graph Laplacian with the Reeb graph to identify and reconstruct the feature curves; and use a variant Cocone algorithm to reconstruct the patch set Ω.

4.2.2 Gaussian-weighted Graph Laplacian

The feature points identification algorithm leverages the widely used Gaussian-weighted graph Laplace operator which we describe in this section. Given a set of points $\boldsymbol{P} = \{\boldsymbol{p}_1, \cdots, \boldsymbol{p}_n\} \subset \mathbb{R}^3$, the (un-normalized) Gaussian-weighted graph Laplacian operator $\mathcal{L}_t$ is an $n \times n$ matrix where

$$\mathcal{L}_t[i][j] = \begin{cases} -\frac{1}{nt^2}\, e^{-\frac{\|\boldsymbol{p}_i - \boldsymbol{p}_j\|^2}{t}}, & \textit{if } i \neq j \\ \frac{1}{nt^2} \sum_{j=1, j\neq i}^{n} e^{-\frac{\|\boldsymbol{p}_i - \boldsymbol{p}_j\|^2}{t}}, & \textit{if } i = j. \end{cases} \tag{4.1}$$

For any fixed function $f : \boldsymbol{P} \to \mathbb{R}$ and a point $\boldsymbol{p}_i \in \boldsymbol{P}$, it is easy to verify that if we apply $\mathcal{L}_t$ to function f, we would get

$$\mathcal{L}_t f(\boldsymbol{p}_i) = \frac{1}{nt^2} \sum_{j=1}^{n} e^{-\frac{\|\boldsymbol{p}_i - \boldsymbol{p}_j\|^2}{t}} \left(f(\boldsymbol{p}_i) - f(\boldsymbol{p}_j) \right)$$

It was shown in [9] that if points in $\boldsymbol{P}$ are uniformly randomly sampled from a smooth d-manifold M, then in the limit as n goes to infinity and t tends to 0 at an appropriate rate, $\mathcal{L}_t f(\boldsymbol{p})$ converges to $\Delta f(\boldsymbol{p})$ where Δ is the Laplace-Beltrami operator for M. The connection to the manifold Laplacian is made via the continuous analog of $\mathcal{L}_t$, the so-called *function Laplacian* [9]:

$$\mathscr{L}_t f(\boldsymbol{p}) = \frac{1}{t^{\frac{d}{2}+1}} \int_M e^{-\frac{\|\mathbf{x} - \boldsymbol{p}\|^2}{t}} \left(f(\boldsymbol{p}) - f(\mathbf{x}) \right) d\mathbf{x} \,.$$

Intuitively, the Gaussian-weighted graph Laplacian $\mathcal{L}_t$ is simply the discretization of the integral operator $\mathscr{L}_t$ at the points in point set $\boldsymbol{P}$. It has been shown that for a sufficiently small t, the integral operator has following property,

$$\mathscr{L}_t f(\boldsymbol{p}) = \Delta f(\boldsymbol{p}) + o(1). \tag{4.2}$$

Now suppose that the underlying domain where data are sampled from is not a manifold, but a singular d-manifold Ω. It turns out that the functional Laplacian $\mathscr{L}_t$ behaves differently around various singularities (i.e., feature curves in 2D) from around a regular point. See [10] for details. In particular, for a point $\mathbf{x}$ lying on a boundary-feature curve of a surface patch Ω_i (i.e., 2-manifold with boundary), we have

$$\mathscr{L}_t f(\mathbf{x}) = \frac{1}{\sqrt{t}} \frac{\pi^{\frac{1}{2}}}{2} \partial_{\mathbf{n}} f(\mathbf{x}) + o(\frac{1}{\sqrt{t}}) \,, \tag{4.3}$$

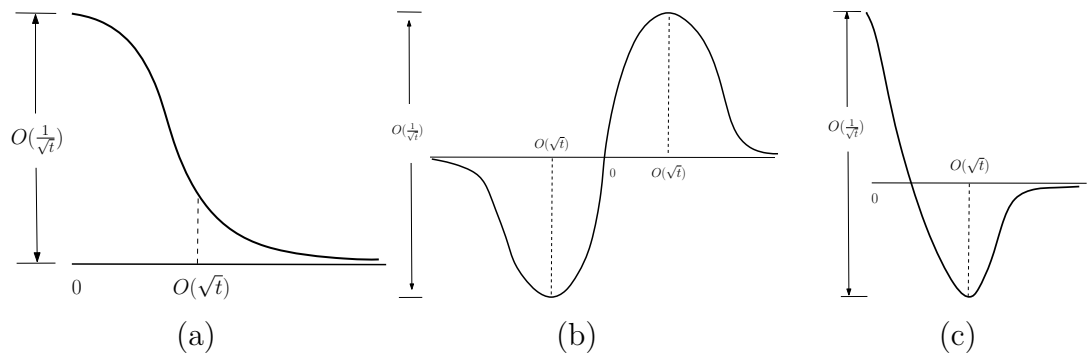


Figure 4.3: The value of the term $O(\frac{1}{\sqrt{t}})$ for points on and around (a) boundary, (b) intersection and (c) sharp corner singularity. In all three figures, 0 is where the singularity locates, x-axis indicates the distance between a point to the singularity, and y-axis indicates the value of the term $O(\frac{1}{\sqrt{t}})$.

where $\mathbf{n}$ is the unit outward normal to the boundary-feature curve at $\mathbf{x}$ (that is, $\mathbf{n}$ is in the tangent space of Ω_i at $\mathbf{x}$ and normal to the boundary-feature curve); and $\partial_{\mathbf{n}} f(\mathbf{x})$ is the directional derivative of f in the direction of $\mathbf{n}$.

Compare Eqn 4.3 with Eqn 4.2, we see the scale dependence on t is different in these two cases: $O(1)$ for a regular point versus $O(\frac{1}{\sqrt{t}})$ for a singular point. For a small t value, which usually is the case in practice, the scale $\frac{1}{\sqrt{t}}$ is large, implying that $\mathscr{L}_t f(\mathbf{x})$ will have a significantly larger value at a boundary point than a regular one (assuming $|\partial_{\mathbf{n}} f(\mathbf{x})|$ is bounded from below by a constant).

The dominance of the $O(\frac{1}{\sqrt{t}})$ term in fact affects a "band" of points within $\Theta(\sqrt{t})$ distance away from the boundary-feature curves. But its value follows a Gaussian distribution centering at the boundary-feature curves with variance t; thus this effect wears off rapidly. See Figure 4.3 for an illustration.

The detailed summary of the behaviour of $\mathscr{L}_t f$ around these feature curves can be found in [29] Appendix.A or [10]. As mentioned earlier, $\mathcal{L}_t$ can be regarded as

a discretization of $\mathscr{L}_t$ and thus shares the similar behaviors as $\mathscr{L}_t$. Our method leverages the different scaling behavior of $\mathcal{L}_t$ around feature curves to help identify feature points in the input point cloud.

4.2.3 Feature curves identification and reconstruction

Now we give a brief introduction of how the Gaussian-weighted graph Laplacian helps us to identify the feature curves hidden in the input point cloud and how to reconstruct such features curves via the Reeb graph strategy we introduced in Chapter 3.

Potential feature points identification. Let the input point cloud be $\boldsymbol{P} = \{\boldsymbol{p}_1, \cdots, \boldsymbol{p}_n\}$. Suppose each point $\boldsymbol{p}_i$ has the coordinate (x, y, z), we apply the graph Laplacian $\mathcal{L}_t$ to the coordinate functions $\mathcal{X}, \mathcal{Y}, \mathcal{Z} : \mathbb{R}^3 \to \mathbb{R}$, where $\mathcal{X}(\boldsymbol{p}_i) = x$, $\mathcal{Y}(\boldsymbol{p}_i) = y$ and $\mathcal{Z}(\boldsymbol{p}_i) = z$.

Now let $\boldsymbol{\mathcal{P}} = [\mathcal{X}, \mathcal{Y}, \mathcal{Z}]$ denote the coordinate functions restricted to the input point set $\boldsymbol{P}$, which is a $n \times 3$ matrix and can be considered as n 3-dimensional (row) vectors. Applying $\mathcal{L}_t$ to $\boldsymbol{\mathcal{P}}$ gives rise to a list of 3-dimensional vectors $\boldsymbol{\mathcal{V}} = \mathcal{L}_t\boldsymbol{\mathcal{P}} = [\mathcal{L}_t\mathcal{X}, \mathcal{L}_t\mathcal{Y}, \mathcal{L}_t\mathcal{Z}]$, where the ith row $\boldsymbol{v}_i = \boldsymbol{\mathcal{V}}[i]$ is a 3-dimensional vector associated with point $\boldsymbol{p}_i \in \boldsymbol{P}$. Also, to simplify the notation, we define another n-dimensional vector V_t, where $V_t[i] = \|\boldsymbol{v}_i\|$ is the normal of vector $\boldsymbol{v}_i$.

Regular surface point. It is known that for a regular surface point $\boldsymbol{p}$ from a smooth surface, its second-order differential quantity has following relation

$$\Delta\boldsymbol{\mathcal{P}}(\boldsymbol{p}) = H_{\boldsymbol{p}} \cdot \mathbf{n}_{\boldsymbol{p}}\,,$$

where $H_{\boldsymbol{p}}$ denotes the mean-curvature at $\boldsymbol{p}$ and $\mathbf{n}_{\boldsymbol{p}}$ is the unit surface normal at $\boldsymbol{p}$. Therefore, we have $\boldsymbol{v}_i = \mathcal{L}_t\boldsymbol{\mathcal{P}}(\boldsymbol{p}_i) \approx H_{\boldsymbol{p}_i} \cdot \mathbf{n}_{\boldsymbol{p}_i}$ at a regular surface point $\boldsymbol{p}_i$.

Point around singular features. However, if point $\boldsymbol{p}_i$ is on or near the three types of aforementioned feature curves, $\boldsymbol{v}_i$ will then reflect fundamentally different information. Specifically, consider a point $\boldsymbol{p}$ on the boundary of a surface path Ω_i. Let $\mathbf{n}$ denote the unit outward normal vector to the boundary curve at $\boldsymbol{p}$. Note, $\mathbf{n} = [n_x, n_y, n_z]$ is not the surface normal at $\boldsymbol{p}$, but the unit vector in the tangent plane at $\boldsymbol{p}$ normal to the boundary curve. By Eqn 4.3, setting $C = \frac{\pi^{1/2}}{2}$, we have

$$\mathscr{L}_t\mathcal{X}(\boldsymbol{p}) \approx \frac{C}{\sqrt{t}}\frac{\partial\mathcal{X}}{\partial\mathbf{n}} = \frac{C}{\sqrt{t}} \cdot \langle[1,0,0,]^T, \mathbf{n}\rangle = \frac{C}{\sqrt{t}}n_x$$

where $[1,0,0,]^T$ is the unit vector in the x-direction, and $\langle .,. \rangle$ stands for the standard inner product of two vectors. By using the approximation $\approx$, we omit the lower order terms $o(\frac{1}{\sqrt{t}})$. Similarly, we can obtain the y and z-direction portion via $\mathscr{L}_t\mathcal{Y}(\boldsymbol{p}) = \frac{C}{\sqrt{t}}n_y$ and $\mathscr{L}_t\mathcal{Z}(\boldsymbol{p}) = \frac{C}{\sqrt{t}}n_z$. Putting them together, we have

$$\mathcal{L}_t\boldsymbol{\mathcal{P}}(\boldsymbol{p}) \approx \mathscr{L}_t\boldsymbol{\mathcal{P}}(\boldsymbol{p}) \approx \frac{C}{\sqrt{t}}[n_x, n_y, n_z]^T = \frac{1}{\sqrt{t}} \cdot \frac{\pi^{1/2}}{2} \cdot \mathbf{n}\,.$$

Note that $\mathcal{L}_t\boldsymbol{\mathcal{P}}(\boldsymbol{p})$ is independent on the choice of the coordinate system, and the magnitude of $\mathcal{L}_t\boldsymbol{\mathcal{P}}(\boldsymbol{p})$ is always $\frac{1}{\sqrt{t}} \cdot \frac{\pi^{1/2}}{2}$, which is usually larger than the magnitude of $H_{\boldsymbol{p}_i} \cdot \mathbf{n}_{\boldsymbol{p}_i}$ at a regular surface point. However, as a point $\boldsymbol{p}$ moves away from the boundary, its magnitude of graph Laplacian operator decreases rapidly from $\frac{1}{\sqrt{t}} \cdot \frac{\pi^{1/2}}{2}$ to the mean curvature (i.e., reduce to be a regular surface point), following the tendency of a Gaussian distribution shape with variance t. At the same time, the direction of $\mathcal{L}_t\boldsymbol{\mathcal{P}}(\boldsymbol{p})$ also changes from the outward normal to the boundary curve to the surface normal for a regular point. Points on and around other type of feature curves share similar behavior.

In other words, $V_t[i]$ is the mean curvature (and thus of the order $O(1)$) for a regular point $\boldsymbol{p}_i$. But $\boldsymbol{v}_i$ is of order $\Theta(\frac{1}{\sqrt{t}})$ for a point on and near feature curves. By "near", we mean points are within $O(c\sqrt{t})$ distance away from the feature curves, where c is a constant whose value depends on the type of singularity we have. Under such setting, the proposed algorithm can easily identify the potential feature curve points as those whose V_t value is above a given threshold τ.

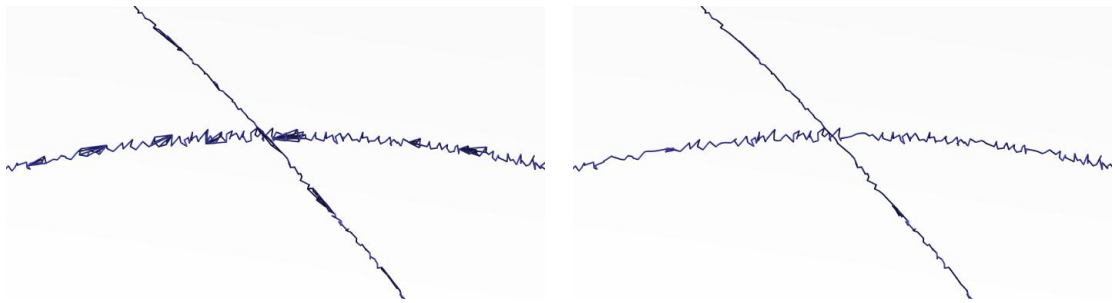
As an example, Figure 4.1 (b) and (c) show the potential feature points identified by our algorithm. Note that as we lower the threshold τ, points with local high mean-curvature will also start to appear as feature points. We remark that as consistent with the theoretical analysis in [10], points "on" intersection-feature curves actually have low $\mathcal{L}_t\boldsymbol{\mathcal{P}}$ magnitude, but points "around" them have high values and are captured as feature points (also shown in Figure 4.3(b)). Hence around an intersection-feature curve, there are two small bands of feature points (see Figure 4.1 (c)). We will see later that our feature curve reconstruction algorithm is able to close the gap between these two bands.

Coarse feature curves reconstruction. Once we identified a set of potential feature points $\boldsymbol{Q}$ from input point clouds $\boldsymbol{P}$, we aim to reconstruct feature curves (graphs) based on them. The subsequent surface reconstruction approaches could be various. Some of them may not explicitly rely on the feature curves. However, in the work we presented in [29], we still need to generate sample points on the feature curves and identify the junction points for the reason we will give soon in Section 4.3 (or see [29] for a thorough description). We achieve this in two steps.

First, we obtain an initial reconstruction of the feature graph Π. This coarse feature graph captures the correct topology of the feature graph that we wish to reconstruct. However, it may not have satisfactory geometry embeddings (such as smooth feature lines aligned with boundaries or sharp features). To this end, we need to further refine the feature graph.

The feature points we obtain are around the feature curves that we wish to reconstruct. We use the algorithm from Chapter 3 (also see [42]) to extract a graph-like structure from input points. In particular, first, we build a simplicial complex K from $\boldsymbol{Q}$ to connect these discrete points. We take K as the Rips complex of $\boldsymbol{Q}$ using radius parameter $\sqrt{t}$; that is, two feature points are connected if their distance is smaller than $\sqrt{t}$, and when the three edges spanned by $p, q, u \in \boldsymbol{Q}$ are in K, we also add the triangle pqu to K (See Figure 1.4, by setting r to be $\frac{1}{2}\sqrt{t}$).

Since all the potential feature points are roughly within a band of width $O(\sqrt{t})$ around the feature curves, we choose $\sqrt{t}$ as the radius parameter to compute the Rips complex. The resulting simplicial complex K "warp" the feature curves (including closing the gap around intersection-feature curves as seen in Figure 4.1 (c) and (d)). Next, we use the algorithm shown in Chapter 3 to compute the Reeb graph of some specific function defined on K to capture the skeleton of the underlying space of K. The output of this graph extraction algorithm is an augmented Reeb graph where each branch is a polygonal curve with vertices being the input feature points in $\boldsymbol{Q}$. The junction nodes where multiple feature curves meet are obtained naturally as graph nodes (i.e., critical points) in the Reeb graph.



(a) before simplification (b) after simplification

Figure 4.4: (a) shows the original Reeb graph with small and noisy loops. (b) shows the same Reeb graph but with those noisy loops removed.

Furthermore, we can easily simplify the resulting feature graphs and remove noisy or less important loops/branches by the simplification process introduced in Chapter 3. See Figure 4.4 as an example where several small spurious loops in the Reeb graph (Figure 4.4(a)) are removed (Figure 4.4(b)).

Feature curve refinement. The feature graphs Π reconstructed above reflect the structures of the hidden feature graphs that we wish to capture. However, the feature curves we obtained are not smooth, and may not be aligned with the real features. In the *feature curve refinement* step, we are trying to improve the quality of the geometry embedding of the feature graphs. There are two components involved. One is for feature graph smoothing, the other is for sharp feature corner location correcting. Figure 4.1(e) shows an example of the the final (refined) feature curve.

1. Smoothing of feature graphs. First, we want to smooth each branch in the feature graph, which is represented as a polygonal curve. A standard curve smoothing algorithm, such as the Laplacian smoothing, tends to push the curve to the center

of the "band" spanned by the feature points involved. Such schema works properly in the intersection curve case but will cause shrinkage for the boundary curves and sharp feature curves. To align them with real features in the process of smoothing, similar to [71], we use an active-contour based approach to deform the feature curve,.

In particular, we design the following energy function:

$$E_{snake} = \sum_{k=1}^{m} \left(E_{int}(\boldsymbol{q}_k) + E_{ext}(\boldsymbol{q}_k) \right), \tag{4.4}$$

where $\boldsymbol{q}_k$ is the k-th vertex in the feature curve which we intend to smooth. The first term $E_{int}(\boldsymbol{q}_k)$ in Eqn 4.4 aims to ensure the smoothness of the feature curve at $\boldsymbol{q}_k$, which is based on approximated derivatives using finite differences and is the same as the one introduced in the original active contour work [62]. The second term $E_{ext}(\boldsymbol{q}_k)$ works to align the actual curves with the real features curves, and is defined as

$$E_{ext}(\boldsymbol{q}_k) = -\sum_{i=1}^{m} e^{-\frac{\|\boldsymbol{p}_i - \boldsymbol{q}_k\|^2}{\sigma^2}} V_t[i], \tag{4.5}$$

where σ is empirically set to be $2\sqrt{t}$. Note that in practice, only the points within $2\sigma = 4\sqrt{t}$ distance from $\boldsymbol{q}_k$ are considered in computing this $E_{ext}(\boldsymbol{q}_k)$.

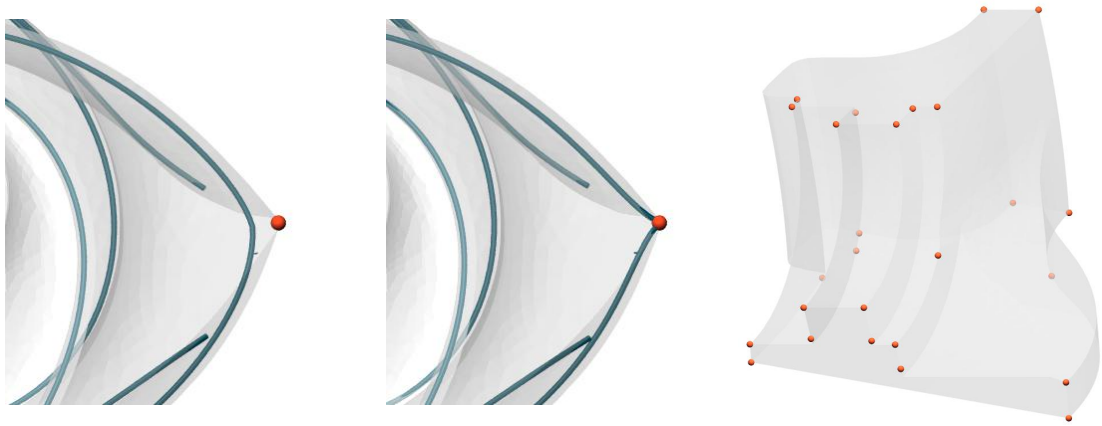
Intuitively, to minimize $E_{ext}(\boldsymbol{q}_k)$, we need to maximize the sum of the Gaussian kernel term, $e^{-\frac{\|\boldsymbol{p}_i - \boldsymbol{q}_k\|^2}{\sigma^2}}$, for point $\boldsymbol{p}_i$ with high V_t values. In other words, $\boldsymbol{q}_k$ should be relocated closer to the points with high V_t. For boundary and sharp feature curves, points with high V_t value lie along the singularities, and this energy term pushes $\boldsymbol{q}_k$ towards them. For the intersection-type of singularity, points with high V_t values are around the intersection lines but not on them (recall Figure 4.1(c) and Figure 4.3(b)). However, if we can assume that the V_t distribution is symmetric along the intersection curve, it is still beneficial to put $\boldsymbol{q}_k$ in the middle of these band of high V_t values, which is coincident with the intersection curves in most cases. While

we cannot prove that this energy function achieves minimum on the singularities, we found that it is effective in practice in handling all three types of singularities within this integrated framework. Another thing to note is that since we have the explicit form for $E_{ext}(\boldsymbol{q}_k)$, we can compute its gradient directly in closed form, which helps to reduce the computational cost.

2. Locations of sharp feature corners. To use the active contour approach described above for refining each individual branch in the feature graph, we need to fix their endpoints (to avoid unwanted shrinkage or shift), which can be represented either by a node where multiple feature curve pieces meet (i.e., graph nodes of degree 3 or more) or a tip of feature curves (i.e., degree-1 graph nodes). We also want to preserve sharp corners within a single feature curve: see Figure 4.5 (a) and (b); such degree-2 corners are not available in the Reeb graph node set (but correspond to the regular points in Reeb graph's arc) and have to be identified separately.

To identify degree-2 sharp corners, we simply compute the local maxima of the function V_t (the magnitude of $\mathcal{L}_t\boldsymbol{\mathcal{P}}$) in the Rips complex K that we constructed to compute the Reeb graph. It turns out that the V_t values are not effective at identifying very sharp corners, due to the limited amount of points available around such corners. To handle this, in spite of the standard weighted graph Laplacian $\mathcal{L}_t$, we also compute its normalized version of the weighted graph Laplacian, $\bar{\mathcal{L}}_t$, and take the local maxima of the magnitude of $\bar{\mathcal{L}}_t\boldsymbol{\mathcal{P}}$.

Corners where 3 or more feature curves meet appear automatically as nodes in the Reeb graph. For each Reeb graph node $\boldsymbol{p}_i$, let $NN(\boldsymbol{p}_i)$ denote its neighbors in the Rips complex K. Our goal is to find a good location for $\boldsymbol{p}_i$ to align with the real



(a) (b) (c)

Figure 4.5: (a) The sharp degree-2 corner is smoothed out after active contour. (b) Our algorithm first identifies degree-2 sharp corners and preserves them during the active contour. (c) Corner points (red dots) computed by our algorithm: they are first identified as nodes of degree≥ 3 in the Reeb graph, and then relocated to align with geometric corners.

sharp geometric corner (0-dimensional singularities) of the hidden domain. Simply taking the centroid of points in $NN(\boldsymbol{p}_i)$ does not serve as a good choice since it tends to push $\boldsymbol{p}_i$ off the domain and away from the sharp corners. So we again take the point with largest V_t value in $NN(\boldsymbol{p}_i)$ as the new position for the corner node $\boldsymbol{p}_i$. See Figure 4.5(c) as an example where the degree-3 nodes are identified by using our algorithm on the "fandisk" model from *Aim@Shape*.

4.3 Feature-Aware Singular Surfaces Reconstruction

So far, we have introduced the algorithm for singular feature curve identification and reconstruction. In some applications, it may require to reconstruct the entire

surfaces instead of merely the feature curves. To this end, one can use a variant of the well known Cocone algorithm [4] for surface reconstruction.

Inspired by the success of ball protection idea of [25] in generating meshes from piecewise smooth complexes, one can use a weighted variant of the Cocone algorithm as proposed in [29] (also see [32]). The original Cocone algorithm filters triangles from the Delaunay triangulation of the input points to reconstruct a smooth surface without boundary. After identifying the feature curves and generating sample points on them in the first step, protecting ball are put centering at each sample point along these curves. The balls are turned into weighted points. These weighted points together with the input points, which remain unweighted, constitute the input to the surface reconstruction algorithm. The Cocone algorithm is run on the weighted Delaunay triangulation of the resulting point set with the constraint that only the unweighted points are allowed to choose the Cocone triangles. This simple modification of the Cocone algorithm allows a quite effective singular surfaces reconstruction as shown in Figure 4.6.

4.4 Remarks

In this chapter, we introduced an algorithm to reconstruct singular surfaces from point could samples. The algorithm can handle boundaries, sharp feature curves, and intersection curves in a unified framework.

The current algorithm can identify the approximate location of corner points where multiple feature curves meet. However, we find it difficult to align these points with the real corner point of the hidden domain when they represent concave corners. A

further investigation to identify the position of these concave corners using only local information would be an interesting direction.

Very recently, some methods other than the protection ball related approaches were proposed for constructing meshes with sharp features, e.g., constructing Isosurfaces using cube merging [12]. It would be interesting to see how to combine the 1D graph feature structure to those algorithms.

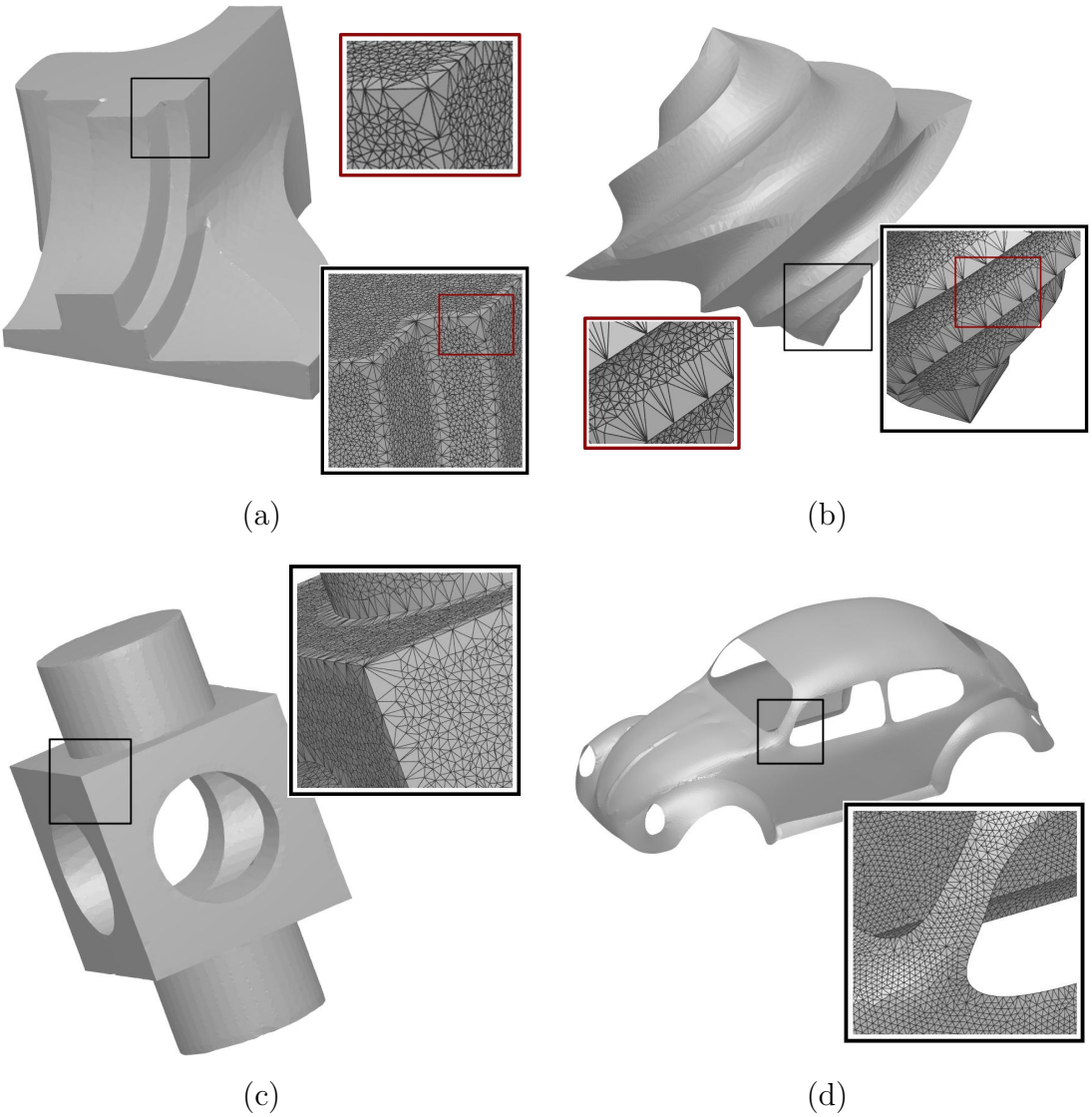


Figure 4.6: Surface reconstruction results for various models.

Chapter 5: Theoretical Bound for Reeb Graph Simplification Distortion

In the previous two chapters, we introduced a Reeb graph based 1D feature structure extracting strategy and demonstrated its effectiveness and usefulness via several applications, including reconstructing singular surfaces from point samples. In the proposed algorithm, a Reeb graph simplification process is used to remove small noisy loops and branches based on their persistence measurement. Such simplification process changes the topology structure of the original Reeb graph. In particular, a prior, it is not clear whether collapsing small features will trigger a domino or cascade effect that causes the killing or significant distortion of large features. In this chapter, we provide a theoretical study on such distortion. Specifically, we will leverage a metric recently proposed in our paper [8], called the functional distortion distance between two Reeb graphs.

5.1 Reeb Graph and Persistent Homology

As mentioned in previous chapters, the Reeb graph of a scalar field on a manifold can be approximated from a point sample set efficiently with theoretical guarantees [30]. It encodes meaningful information on the input scalar field; and being a graph structure, it is simple to represent and manipulate.

Reeb Graph. The basic concept of Reeb graph has been given in Chapter 1.2.1 , so we will not repeat it here. Instead, we introduce an alternative view of the definition in terms of the quotient space. First we define an equivalence relation $\sim$ on the topological space X such that $x \sim y$ if and only if $f(x) = f(y) = \alpha$ and x is connected to y in the level set $f^{-1}(\alpha)$. The *Reeb space* of the function $f : X \to \mathbb{R}$, denoted by $\mathcal{R}_f$, is the quotient space $X/\sim$., i.e., the set of equivalent classes equipped with the quotient topology induced by the quotient map $\mu : X \to \mathcal{R}_f$. Under appropriate regularity assumptions, $\mathcal{R}_f$ has the structure of a finite 1-dimensional regular CW complex, and we call it a *Reeb graph*. (Throughout this chapter, we assume that all mentioned connected components are also path-connected.)

The input function $f : X \to \mathbb{R}$ also induces a continuous function from Reeb graph to the scalar field $\widetilde{f} : \mathcal{R}_f \to \mathbb{R}$ defined as $\widetilde{f}(z) = f(x)$ for any preimage $x \in \mu^{-1}(z)$ of z. We following write $\widetilde{f}(z)$ as $f(z)$ for $z \in \mathbb{R}_f$ to simplify the notation. For all the illustrations in this chapter, if not mention specifically, we plot the Reeb graph with the vertical coordinate of a point z corresponding to the function value $f(z)$. That is, one can assume the function f is always the height field.

Given a point $x \in \mathcal{R}_f$, we use the term *up-degree* (resp. *down-degree*) of x to denote the number of branches (1-cells) incident to x that have higher (resp. lower) values of f than x. A point is called *regular point* if both of its up-degree and down-degree equal to 1; otherwise, it's a *critical point*. A critical point is a minimum (maximum) if it has down-degree 0 (up-degree 0), and a down-fork (up-fork) if it has down-degree (up-degree) larger than 1. In some literature, down-fork and up-fork are also referred as down-saddle and up-saddle or down-fork saddle and up-fork saddle, respectively. A critical point can be degenerate, that is, it can have more than one

types of criticality. From now on, we use the term *node* to refer to a critical point in the Reeb graph. For simplicity of exposition, we assume that all nodes of the Reeb graph have distinct $\widetilde{f}$ function values.

Persistent homology and persistence diagrams. The notion of persistence was originally introduced by Edelsbrunner et al. in [38]. Since then, there has been a great amount of developments both in theory and in applications [87, 15, 19, 31]. Here we do not concern the theory of persistence but merely use it to describe the graph "features" and their "importance". Hence, as follows, we only provide a simple description so as to introduce the notion of persistence diagrams.

We first introduce the concept of sublevel set and suplevel set, which is analogous to the level set concept shown in Chapter 1.2.1. Given a function $f : X \to \mathbb{R}$ on a topological space, we call the topology space $X_{\leq a} = \{x \in X | f(x) \leq a\}$ the *sublevel set* of X w.r.t. f. The corresponding concept of *suplevel set* can be defined as $X_{\geq a} = \{x \in X | f(x) \geq a\}$. As we sweep through X in increasing value of a, we inspect the changes in $\mathrm{H}_p(X_{\leq a})$, where we use $\mathrm{H}_p(X)$ to denote the p-th homology group of topology space X. In this procedure, some new homology classes such as new components, which give 0-dimensional homology classes, or loops, which give 1-dimensional classes, will be created; also, existing homology classes can be destroyed (e.g. two components merge to be a single one). Persistent homology records such birth and death events, and the corresponding information will be encoded in persistence diagram.

For Reeb graph $\mathcal{R}_f$, there is a natural way to define "features" and rank their "importance". That way turns out to be consistent with the persistence diagram of function $f : \mathcal{R}_f \to \mathbb{R}$. Since $\mathcal{R}_f$ is a graph, we only need to consider persistent

homology in dimensions 0 and 1. We provide an intuitive treatment below. For simplicity of exposition, we assume that all nodes have different function values and are either a minimum, a maximum, a down-fork with down-degree 2, or an up-fork with up-degree 2. Note that these assumptions hold in the generic case.

Imagine that we sweep through $\mathcal{R}_f$ in increasing values of a and inspect changes in $\mathrm{H}_0\big((\mathcal{R}_f)_{\leq a}\big)$. New components in the sublevel sets are created at minima of $\mathcal{R}_f$. For any value a, associate each component C in the sublevel set of $(\mathcal{R}_f)_{\leq a}$ with the lowest local minimum m contained in C: intuitively, C is created at m.

Consider a down-fork node s with $f(s) = a$. If the two lower branches are contained in different connected components C_1 and C_2 of the open sublevel set $(\mathcal{R}_f)_{<a}$, we call s an *ordinary fork*; otherwise, it is an *essential fork*. Let v_1 and v_2 be the global minimum of components C_1 and C_2, respectively. Assume that $f(v_1) < f(v_2)$. The homology class $[v_1 + v_2]$ is created at v_2, which is the minimum of the "younger" component, and dies at down-fork node s, giving rise to a unique point $\big(f(v_2), f(s)\big)$ in the 0-th ordinary persistence diagram $D_{\mathrm{g}_0}(\mathcal{R}_f)$. There is a one-to-one correspondence between the set of such pairs of minima and ordinary down-fork and points in the 0th persistence diagram $D_{\mathrm{g}_0}(\mathcal{R}_f)$ with finite coordinates. See Figure 5.1 and Figure 5.2(left) as an example. A symmetric procedure with $-f$ will produce pairs of maxima and ordinary up-forks, which correspond to the points in the 0th persistence diagram $D_{\mathrm{g}_0}(\mathcal{R}_{-f})$. Together, these pairs capture the *branching features* of a Reeb graph.

On the other hand, if the two lower branches of s have been connected in the sublevel set, we call s an *essential fork*; see Figure 5.1 and Figure 5.2(right). In such case, a set of 1-cycle in the sublevel set $(\mathcal{R}_f)_{\leq f(s)}$ are/is born at s. Since $\mathcal{R}_f$ is a

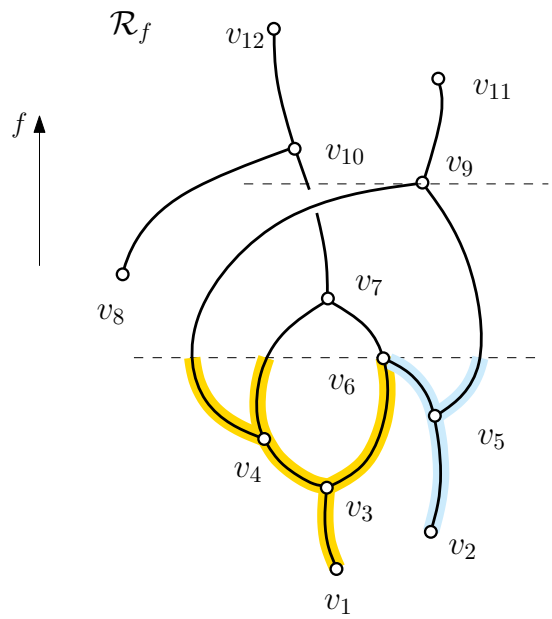


Figure 5.1: In above Reeb graph $\mathcal{R}_f$, nodes v_1, v_2 and v_8 are minimum; v_{11} and v_{12} are maximum. v_3, v_4 and v_5 are up-fork nodes and the rest are the down fork nodes. The down fork v_6 (*original saddle*) merges components C_1 (highlighted in yellow) and C_2 (highlighted in light blue) in the sublevel set below it, represented by minima v_1 and v_2, respectively. The down fork v_9 (an *essential saddle*) is paired with the up fork v_4, corresponding to the thin loop/cycle $v_4v_9v_5v_6v_7v_4$.

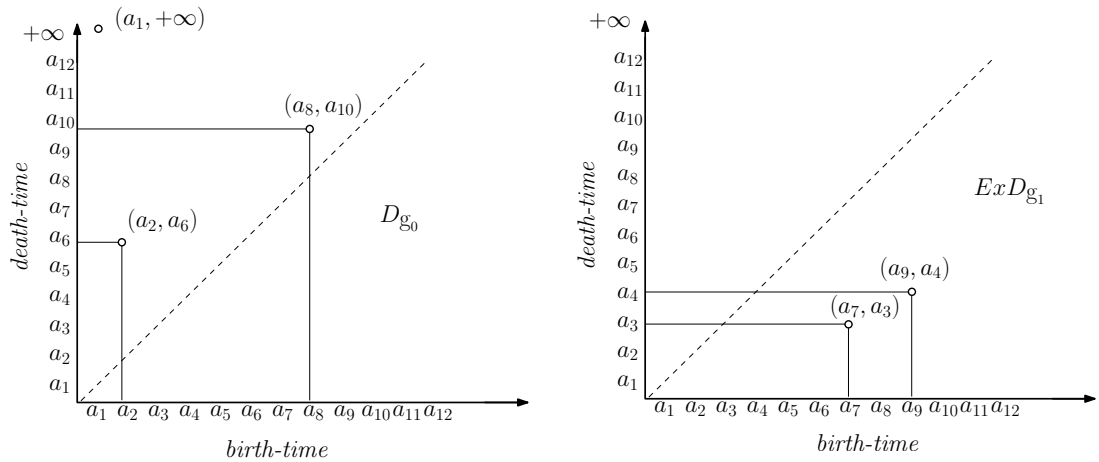


Figure 5.2: The critical pair (v_2, v_6) generated in Figure 5.1 gives rise to the point (a_2, a_6) in the ordinary persistence diagram, D_{g_0} (left), where $a_i = f(v_i)$ for $i = 1, \cdots, 12$. The essential fork v_9 is paired with the up-fork v_4, corresponding to the thin loop $v_4v_9v_5v_6v_7v_4$ created at v_9. This gives rise to the point (a_9, v_4) in the extended persistence diagram ExD_{g_1} (right).

graph, such cycles are non-trivial in $\mathcal{R}_f$, and their corresponding homology classes will not be destroyed in ordinary persistent homology. Consider the unique cycle γ with largest minimum value of f among all cycles born at s and corresponding to an embedded loop in $\mathcal{R}_f$. Let s' be the point achieving the minimum on γ. Then we say that cycle γ is created at $f(s)$ and killed at time $f(s')$ in the extended part. This gives rise to a unique point $\big(f(s), f(s')\big)$ in the 1st extended persistence diagram of f. It turns out that s' is necessarily an essential up-fork node [3], and we call such a pair (s', s) an *essential pair*. Indeed, the collection of essential pairs has a one-to-one correspondence to points in $ExD_{g_1}(\mathcal{R}_f)$. Note, the extended persistence diagram $ExD_{g_1}(\mathcal{R}_{-f})$ is the reflection of $ExD_{g_1}(\mathcal{R}_f)$ and thus encodes the same information as $ExD_{g_1}(\mathcal{R}_f)$. These essential pairs capture the cycle features of a Reeb graph.

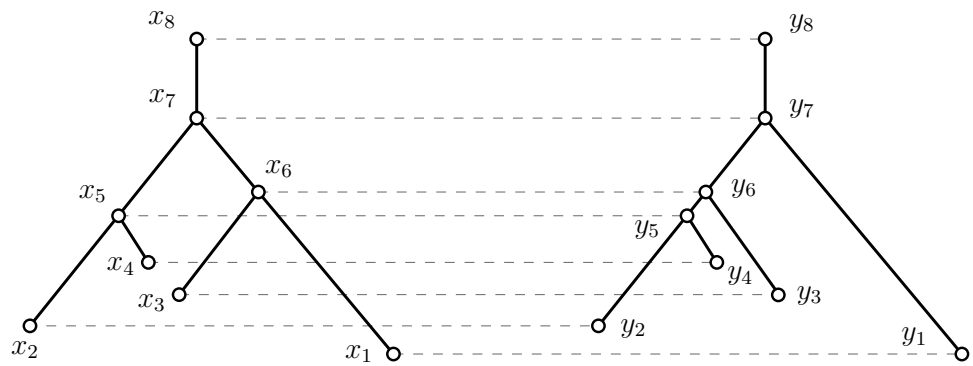


Figure 5.3: The height functions on the two trees have the same persistence diagrams (thus the bottleneck distance between their persistence diagrams is 0), but their tree structures are different. The functional distortion distance will differentiate these two cases.

In short, the branching features and cycle features of a Reeb graph give rise to the points in the 0th ordinary and 1st extended persistence diagrams, respectively. In order to measure changes in persistence diagram for two different Reeb graphs, it turns out that we can use a concept called functional distortion distance.

5.2 A Metric on Reeb graph

Given the popularity of the Reeb graph in data analysis, it is important to understand its stability and robustness with respect to changes in the input function (both in function values and in the domain). To measure the stability, we first need to define a distance between two Reeb graphs. To this end, we propose a metric for Reeb graphs, called the *functional distortion distance*. Under this distance, the Reeb graph is stable against perturbations of the input function; at the same time, it retains a certain ability to discriminate between different functions.

5.2.1 Distance between two points in graph

In the remaining of this chapter, by a *distance*, we mean an extended pseudometric, i.e., a binary symmetric function $d : X \times X \to \mathbb{R}_{\geq 0}$ s.t., $\forall x, y, z \in X$

(1) $d(x, x) = 0$

(2) $d(x, z) \leq d(x, y) + d(y, z)$

From now on, we will consider two Reeb graphs $\mathcal{R}_f$ and $\mathcal{R}_g$, generated by functions $f : X \to \mathbb{R}$ and $g : X \to \mathbb{R}$, respectively. While topologically each Reeb graph is simply a 1-dimensional regular CW complex, it is important to note that it also has a function associated with it (induced from the input scalar field). Hence the distance should depend on both the graph structures and the functions $\widetilde{f}$ and $\widetilde{g}$. Approaching the problem through graph isomorphisms does not seem viable, as small perturbation of the function f may create an arbitrary number of new branches and loops in the graph. To this end, we first put the following metric structure on a Reeb graph $\mathcal{R}_f$ to capture information about the function f .

Specifically, for any two points $u, v \in \mathcal{R}_f$ (not necessarily to be graph nodes), let π be a continuous path between u and v. The *range* of this path is the interval

$$\text{range}(\pi) := [\min_{x \in \pi} f(x), \max_{x \in \pi} f(x)] \,,$$

and its *height* is simply the length of the range, denoted by

$$\text{height}(\pi) = \max_{x \in \pi} f(x) - \min_{x \in \pi} f(x) \,.$$

We then define the distance as

$$d_f(u, v) = \min_{\pi : u \rightsquigarrow v} \text{height}(\pi) \,, \tag{5.1}$$

where π ranges over all paths from u to v, denoted by $u \rightsquigarrow v$. Equivalently, $d_f(u,v)$ is the minimum length of any interval I such that u and v are in the same component of $f^{-1}(I)$. Note that this is in fact a metric, since on Reeb graphs there is no path of constant function value between two points $u \neq v$. Intuitively, $d_f(u,v)$ can be regarded as the minimal function difference one has to overcome to move from u to v.

5.2.2 Functional distortion distance d_{FD}

We can view the Reeb graphs $\mathcal{R}_f$ and $\mathcal{R}_g$ as metric graphs $\mathscr{R}_f = (\mathcal{R}_f, d_f)$ and $\mathscr{R}_g = (\mathcal{R}_g, d_g)$, equipped with metrics d_f and d_g, respectively. A natural distance for metric spaces is called *Gromov-Hausdorff* distance (GH-distance), which we define below, following the definitions and results from [64].

Definition 5.2.1 (Correspondence [64]) *For set A and B, a subset $C \subset A \times B$ is a correspondence between A and B, if and only if,*

(1) $\forall\, a \in A$, there exists $b \in B$ s.t. $(a,b) \in C$;

(2) $\forall\, b \in B$, there exists $a \in A$ s.t. $(a,b) \in C$.

Let $\Lambda(A,B)$ denote the set of all possible correspondences between sets A and B.

Definition 5.2.2 (Gromov-Hausdorff Distance [64]) *The Gromov-Hausdorff distance between two metric spaces $\mathscr{X} = (X, d_X)$ and $\mathscr{Y} = (Y, d_Y)$ is*

$$d_{GH}(\mathscr{X}, \mathscr{Y}) = \frac{1}{2} \inf_{C \in \Lambda(A,B)} \max_{(x,y),(x',y') \in C} |d_X(x,x') - d_Y(y,y')| \,.$$

We now can talk about the Gromov-Hausdorff distance between metric graphs $\mathcal{R}_f$ and $\mathcal{R}_g$ using the distance we defined in 5.2.1. To do so, we first connect the space

$\mathcal{R}_f$ and $\mathcal{R}_g$, which is achieved by continuous maps $\phi : \mathcal{R}_f \to \mathcal{R}_g$ and $\psi : \mathcal{R}_g \to \mathcal{R}_f$. Then we borrow above GH-distance definition (or see [51]) and let

$$G(\phi, \psi) = \big\{(x, \phi(x)) : x \in \mathcal{R}_f\big\} \cup \big\{(\psi(y), y) : y \in \mathcal{R}_g\big\} \quad \text{and}$$
$$D(\phi, \psi) = \sup_{(x,y),(\tilde{x},\tilde{y}) \in G(\phi,\psi)} \frac{1}{2} \left| d_f(x, \tilde{x}) - d_g(y, \tilde{y}) \right|, \tag{5.2}$$

where $G(\phi, \psi)$, the union of the graphs of ϕ and ψ, can be thought of as the set of correspondences between $\mathcal{R}_f$ and $\mathcal{R}_g$ induced by maps ϕ and ψ. The *functional distortion distance* is defined as:

$$d_{FD}(\mathcal{R}_f, \mathcal{R}_g) = \inf_{\phi,\psi} \max \big\{D(\phi, \psi), \|f - g \circ \phi\|_\infty, \|f \circ \psi - g\|_\infty\big\}, \tag{5.3}$$

where ϕ and ψ range over all continuous maps between $\mathcal{R}_f$ and $\mathcal{R}_g$. The latter two terms address the fact that composition with isometries of the real line (translation, negation) does not affect the metric d_f induced by a function f. Note that this definition can be considered as a continuous, functional variant of the Gromov–Hausdorff distance, with the additional condition that the maps between $\mathcal{R}_f$ and $\mathcal{R}_g$ are required to be continuous, and taking into consideration the difference between the function values of corresponding points as well. In fact, this definition is the continuous version of the extended Gromov-Hausdorff distance introduced in Definition 2.4 of [20]. Furthermore, it turns out that for metric graphs, our continuous version of the extended Gromov-Hausdorff (GH) distance is a constant factor approximation of the extended GH distance induced by arbitrary maps. As an example, consider the two trees in Figure 5.3. The distortion of distances in the two trees in Figure 5.3(left) is large no matter how we identify correspondences between points from them. Thus

the functional distortion distance between them is also large, making it more discriminative than the bottleneck distance between persistence diagrams, which is 0 in this two-tree case .

It is straightforward to show that the functional distortion distance is a pseudometric, and a metric on the equivalence classes of Reeb graphs up to function-preserving homeomorphisms. Note that this definition and our results apply to any graph G with a function f that is strictly monotonic on the edges. This is easy to see since in that case $\mathcal{R}_f = G$ and $\widetilde{f} = f$.

5.2.3 Relation to Bottleneck Distance and Gromov-Hausdorff Distance

1. Relation to Bottleneck Distance. We have seen in Figure 5.3 that there are cases where the functional distortion distance can be strictly larger (more discriminative) than the bottleneck distance between persistence diagrams of according dimensions (0th ordinary and 1st extended persistent diagrams). In addition, as shown in Theorem 5.2.3 and 5.2.4, the functional distortion distance is always at least as large as (as discriminative as) the bottleneck distance. For the branching features (ordinary persistence diagram), we have

Theorem 5.2.3 $d_B(D_{g_0}(\mathcal{R}_f), D_{g_0}(\mathcal{R}_g)) \leq d_{FD}(\mathcal{R}_f, \mathcal{R}_g)$.
Similarly, $d_B(D_{g_0}(\mathcal{R}_{-f}), D_{g_0}(\mathcal{R}_{-g})) \leq d_{FD}(\mathcal{R}_f, \mathcal{R}_g)$.

For the cycle features (extended persistence diagram), we have the distance relation.

Theorem 5.2.4 $d_B(ExD_{g_1}(\mathcal{R}_f), ExD_{g_1}(\mathcal{R}_g)) \leq 3d_{FD}(\mathcal{R}_f, \mathcal{R}_g)$.

The proof of both theorems can be found in [7] Appendix.B.

2. Relation to Gromov-Hausdorff Distance. We can view the Reeb graphs $\mathcal{R}_f$ and $\mathcal{R}_g$ as metric spaces, equipped with metrics d_f and d_g, respectively. A natural distance for metric spaces is the *Gromov–Hausdorff* distance, which is defined as follows, using the notation of Eqn 5.2:

$$d_{GH}(\mathcal{R}_f, \mathcal{R}_g) = \inf_{\phi,\psi} \left(D(\phi, \psi) \right), \tag{5.4}$$

where $\phi : \mathcal{R}_f \to \mathcal{R}_g$ and $\psi : \mathcal{R}_g \to \mathcal{R}_f$ are all *maps* between $\mathcal{R}_f$ and $\mathcal{R}_g$. Here the maps ϕ, ψ are not necessarily *continuous*, which is different from our definition of the functional distortion distance.

Note that translation $f + c$ and negation $-f$ do not change the metric structures of the Reeb graph $\mathcal{R}_f$. To remove the effect of the difference in the function values, we define the *functional Gromov-Hausdorff distance* between $\mathcal{R}_f$ and $\mathcal{R}_g$, which measures not only the distance distortion, but also the changes in the function value between corresponding points:

$$d_{fGH}(\mathcal{R}_f, \mathcal{R}_g) := \inf_{\phi,\psi} \max \left(D(\phi, \psi), \|f - g \circ \psi\|_\infty, \|f \circ \phi - g\|_\infty \right), \tag{5.5}$$

where ϕ and ψ range over all maps between $\mathcal{R}_f$ and $\mathcal{R}_g$.

It turns out that we have the following relations, which imply that our functional distortion distanceroughly measures the minimum distortion in both function values (between f and g) and in their induced metrics (between d_f to d_g). We note that this result holds if we remove the function value differences (i.e., $\|f - g \circ \phi\|_\infty$ and $\|f \circ \psi - g\|_\infty$) from both d_{fGH} and d_{FD}. That is, the continuous version of Eqn 5.4 approximates the standard Gromov-Hausdorff distance for metrics d_f and d_g. We also note that this relation does not generalize to spaces with dimension higher than one.

Theorem 5.2.5 $d_{fGH}(\mathcal{R}_f, \mathcal{R}_g) \leq d_{FD}(\mathcal{R}_f, \mathcal{R}_g) \leq 3d_{fGH}(\mathcal{R}_f, \mathcal{R}_g)$.

We do not include the proof of Theorem 5.2.5 in this dissertation work. One may refer to [7] Appendix.A for details if interested.

5.3 Simplification of Reeb graphs

As we described in Section 5.1, there is a natural way to quantify branching and loop features in terms of ordinary and extended persistence in the according dimensions. The persistence based simplification can help to remove noisy or less important features. Also it can be used to create a multi-resolution representation of the input domain as shown in previous chapters. In this section, we prove that by removing small features using a natural merging strategy, (branching and loop) features with large persistence value will not be killed, in other words, their persistence, i.e., "importance", will be roughly maintained.

5.3.1 A natural simplification scheme for Reeb graph

We first introduce a merging based simplification strategy for Reeb graphs. The strategy has been used and roughly mentioned in Chapter 3 and Chapter 4.

Branching feature. Given an ordinary persistence pair (m, s) with m as a minimum and s as a down-fork node. Recall that the down-fork s merges two disconnected components C_1 and C_2 of the sublevel set below $f(s)$, and m of the one with higher f value (in the minimum of C_1 and C_2). To remove the feature (m, s), we wish to merge the branch containing m, say C_2, into the other branch C_1, so that afterwards, m and s become regular points (i.e, with up-degree and down-degree both equal to 1). In particular, we perform the following operations (as shown in Figure 5.4). In

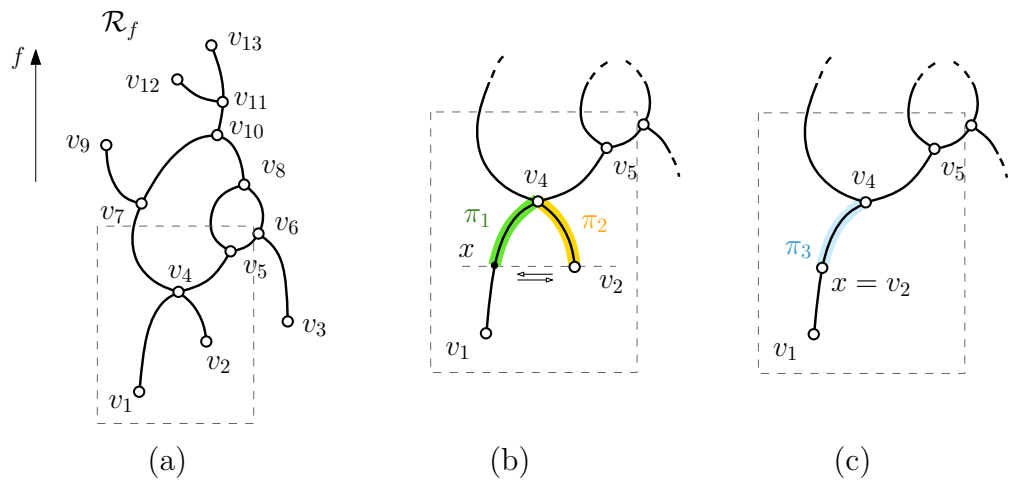


Figure 5.4: (a) shows the original Reeb graph before simplification (also see Figure 1.5). We take the branches inside the dashed box as an example. (b) shows the paths to be merged, i.e., π_1 (green) and π_2 (yellow). (c) shows the merged path, i.e., a monotonic path, denoted by π_3 (blue). The simplification result for the entire Reeb graph $\mathcal{R}_f$ can be found in Figure 1.5(c).

Figure 5.4, let v_1 and v_2 denote the minimum of component $C1$ and C_2, respectively. We choose an arbitrarily embedded path $\pi_2 \subset C_2$ from v_4 to v_2, and an arbitrary path π' from v_4 to v_1. Now imagine that starting from v_4 we traverse the path π' downwards. We stop when we encounter the first point $x \subset C_1$, such that $f(x) = f(v_2)$, and set π_1 to be the subcurve of π' from v_4 to x. By identifying points with same function value, we merge $\operatorname{im}\pi_1$ and $\operatorname{im}\pi_2$ to form the image of a new monotonic arc π_3 between v_4 (i.e., the down-fork node) and x such that any point $p \in \operatorname{im}\pi_1 \cup \operatorname{im}\pi_2$ is mapped to some $q \in \pi_3$ with $f(p) = f(q)$. Pairs of up-fork and maximum are treated in a symmetric way.

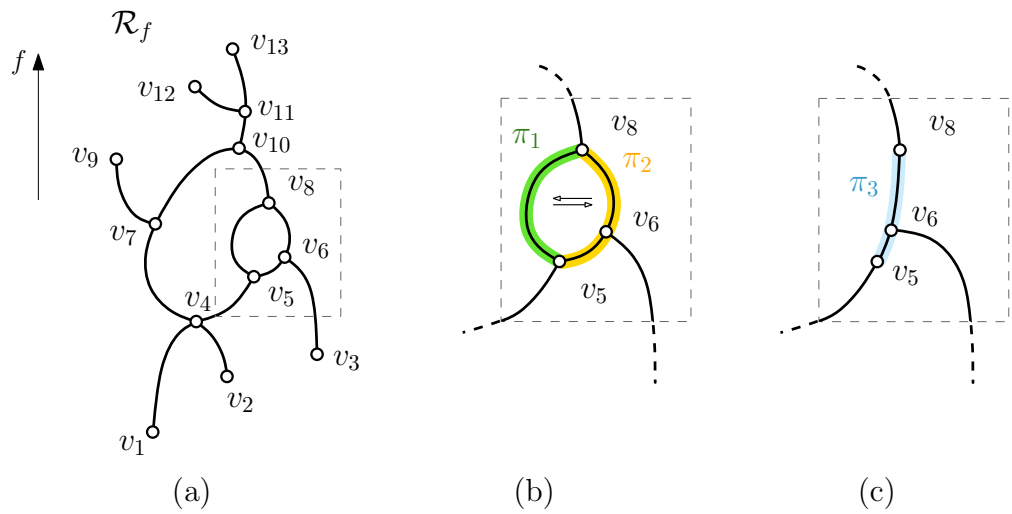


Figure 5.5: (a) shows the original Reeb graph before simplification (also see Figure 1.5). We take the small loop inside the dashed box as an example. (b) shows the paths (two half cycles) to be merged, i.e., π_1 (green) and π_2 (yellow). (c) shows the merged cycle. Now it's a monotonic path denoted by π_3 (blue). The simplification result for the entire Reeb graph $\mathcal{R}_f$ can be found in Figure 1.5(c).

Cycle feature. Given an extended persistence pair (s_1, s_2) between an up-fork s_1 and a down-fork s_2, let γ be a thin cycle[4] spanned by these two nodes. W.l.o.g. assume that $\operatorname{im}\gamma$ only contains one single connected component: if $\operatorname{im}\gamma$ has multiple connected components, then there must exist one that contains both s_1 and s_2. That component is necessarily an embedded loop and thus we can simply set γ to be the thin cycle corresponding to that loop. Let π_1 and π_2 denote the two disjoint subcurves of the loop that connect s_1 and s_2. To cancel the feature, intuitively, we wish to merge π_1 and π_2 so that the cycle γ will be destroyed. Note that π_1 and π_2 may not be monotonic (w.r.t. the input function f); however, all points in π_1 and π_2 have function values within the range $[f(s_1), f(s_2)]$. The merging of π_1 and π_2 results in a

[4]A cycle γ is *thin* if there is no other cycle with smaller height than γ but having the same minimum or maximum value in f as γ

new monotonic arc π_3 from s_1 and s_2, such that every point $p \in \operatorname{im}\gamma$ is mapped to some $q \in \pi_3$ with $f(q) = f(p)$. See Figure 5.5 for an illustration, where the down-fork is v_8 and the up fork is v_5.

Note that since a critical pair (m, s) (resp. an essential pair (s_1, s_2)) corresponds uniquely to a persistence pair $(f(m), f(s))$ in the ordinary persistence diagram (resp. $(f(s_1), f(s_2))$ in the extended persistence diagram), the above process also removes a point from the respective persistence diagram.

Let $\mathcal{R}$ and $\mathcal{R}'$ denote the Reeb graph before and after the simplification of a persistence pair $\tau = (b, d)$ by collapsing its corresponding branching or loop feature. Let π_1^τ and π_2^τ be as introduced above. Call $\gamma^\tau = \pi_1^\tau \cup \pi_2^\tau$ the *merging path w.r.t.* τ. Note that γ^τ is a closed curve corresponding to a thin cycle spanning (b, d) when it is an extended persistence pair, and a connected path with b and d being the respective minimum and maximum function values on it otherwise. The merging path γ^τ will be collapsed into a single monotonic arc in order to eliminate the persistence pair τ. We can view the removal of τ in a more formal way as follows: We say that two points $x, y \in \mathcal{R}$ are τ-equivalent, denoted by $x \sim_\tau y$, if $f(x) = f(y)$ and $x, y \in \gamma^\tau$. The simplified Reeb graph $\mathcal{R}'$ is the quotient space $\mathcal{R}/\sim_\tau$; the corresponding quotient map $\mu_\tau : \mathcal{R} \to \mathcal{R}'$ satisfies $\mu_\tau(x) = \mu_\tau(y)$ if and only if $x \sim_\tau y$. The function $f : \mathcal{R} \to \mathbb{R}$ induces a function $f' : \mathcal{R}' \to \mathbb{R}$ such that for any $x' \in \mathcal{R}'$, $f'(x') = f(x)$ for any $x \in \mu_\tau^{-1}(x')$.

Now given an input Reeb graph $\mathcal{R}$, suppose we wish to eliminate a set of persistence pairs $\{\tau_1 = (b_1, d_1), \tau_2 = (b_2, d_2), \ldots, \tau_k = (b_k, d_k)\}$. Compute the merging path γ^{τ_i} for each persistence pair τ_i in $\mathcal{R}$. We now define an equivalence relation $\sim$ as the transitive closure of all $\sim_{\tau_i}$s for $i \in [1, k]$. This is equivalent to collapsing γ^{τ_i}s

for all $i = 1, \cdots, k$ in an arbitrary order to kill the persistence pairs $\tau_1, \ldots, \tau_k$ The final simplified Reeb graph $\widetilde{\mathcal{R}}$ is obtained as the quotient space $\mathcal{R}/\sim$. We have a well-defined function g that

$$\begin{aligned} g : \widetilde{\mathcal{R}} \to \mathbb{R} \text{ induced by the function } f : \mathcal{R} \to \mathbb{R} \\ \text{s.t. } g(\mu(x)) = f(x),\ x \in \mathcal{R}. \end{aligned} \tag{5.6}$$

Let δ denote the largest persistence of $\tau_1, \ldots, \tau_k$. We have the following properties of $\widetilde{\mathcal{R}}$, whose proof can be found in [8] Appendix.D.

Observation 5.3.1 *(i) Given any two points $x, y \in \mathcal{R}$, we have $d_g(\mu(x), \mu(y)) \leq d_f(x, y)$.*

(ii) Given a point $\tilde{x} \in \widetilde{\mathcal{R}}$, for any two points $x_0, x_1 \in \mu^{-1}(\tilde{x})$, we have $d_f(x_0, x_1) \leq 2\delta$.

5.3.2 Distance between $\mathcal{R}$ and $\widetilde{\mathcal{R}}$

While the simplification scheme removes persistence pairs $\tau_1, \ldots, \tau_k$, it is not clear how other points in the persistence diagram of the original Reeb graph $\mathcal{R}$ are affected. In this section, we bound the bottleneck distance between the persistence diagrams of $\mathcal{R}$ and $\widetilde{\mathcal{R}}$. Specifically, we bound the functional distortion distance, $d_{FD}(\mathcal{R}, \widetilde{\mathcal{R}})$, where we have $f : \mathcal{R} \to \mathbb{R}$ and $g : \widetilde{\mathcal{R}} :\to \mathbb{R}$ (defined in Eqn 5.6 Section 5.3.1). We do so by constructing continuous maps $\phi : \mathcal{R} \to \widetilde{\mathcal{R}}$ and $\psi : \widetilde{\mathcal{R}} \to \mathcal{R}$ as we did in Section 5.2.2, and bounding the terms in Eqn 5.3, which in turn provides an upper bound for $d_{FD}(\mathcal{R}, \widetilde{\mathcal{R}})$. Note, by the construction of quotient map μ for the simplification process and the definition of surjective mapping ϕ, we see $\phi = \mu$.

Construct map ψ_ε. The continuous map $\phi : \mathcal{R} \to \widetilde{\mathcal{R}}$ can simply be taken as the surjective map $\varphi : \mathcal{R} \to \widetilde{\mathcal{R}}$. For the opposite direction, we will construct a sequence of

maps $\psi_\varepsilon : \widetilde{\mathcal{R}} \to \mathcal{R}$. First, we need the following result, which is a slight generalization of Observation 5.3.1.

Lemma 5.3.2 *Let $\tilde{x}, \tilde{y} \in \widetilde{\mathcal{R}}$ be two points in $\widetilde{\mathcal{R}}$ such that there exists a monotonic path $\tilde{\pi}$ between $\tilde{x}$ and $\tilde{y}$ with $d_g(\tilde{x}, \tilde{y}) = g(\tilde{y}) - g(\tilde{x}) = \varepsilon$. Let x and y be arbitrary preimages for $\tilde{x}$ and $\tilde{y}$, respectively. Then $d_f(x, y) \leq 2\delta + \varepsilon$.*

In fact, there is a path π from x to y such that the highest point t in $\operatorname{im}\pi$ satisfies $f(t) \leq f(y) + \delta$, and the lowest point b in $\operatorname{im}\pi$ satisfies $f(b) \geq f(x) - \delta$.

Now for a fixed positive real ε, we use the following procedure to construct a continuous map $\psi^\varepsilon : \widetilde{\mathcal{R}} \to \mathcal{R}$. First, we subdivide the simplified Reeb graph $\widetilde{\mathcal{R}}$ by adding a set of nodes, so that every arc in the resulting graph (still denoted by $\widetilde{\mathcal{R}}$) has height at most ε. Note that the height of an monotonic path from x to y is simply the difference in the function values of x and y. We refer to the resulting augmented graph $\widetilde{\mathcal{R}}$ as *an ε-subdivision of* $\widetilde{\mathcal{R}}$ with nodes $V_\varepsilon = \{\tilde{v}_1, \ldots \tilde{v}_m\}$. Now for each $\tilde{v}_i \in V_\varepsilon$, we set $\psi^\varepsilon(\tilde{v}_i)$ to be an arbitrary but fixed pre-image $v_i \in \mu^{-1}(\tilde{v}_i)$. For each arc $\tilde{\pi}(\tilde{v}_i, \tilde{v}_j)$ of $\widetilde{\mathcal{R}}$, consider the path $\pi(v_i, v_j)$ connecting the two preimage points v_i and v_j as stated in Lemma 5.3.2. We set the restriction of ψ^ε over the arc $\tilde{\pi}(\tilde{v}_i, \tilde{v}_j)$ to be any homeomorphism from $\tilde{\pi}(\tilde{v}_i, \tilde{v}_j)$ to $\pi(v_i, v_j)$ with $\psi^\varepsilon(\tilde{v}_i) = v_i$ and $\psi^\varepsilon(\tilde{v}_j) = v_j$. The maps $\psi^\varepsilon(\tilde{\pi}(\tilde{v}_i, \tilde{v}_j))$ for all arcs $\tilde{\pi}(\tilde{v}_i, \tilde{v}_j)$ of $\widetilde{\mathcal{R}}$ assemble to the continuous map $\psi^\varepsilon : \widetilde{\mathcal{R}} \to \mathcal{R}$.

By definition of $\phi = \mu$, we have $\max_{x \in \mathcal{R}} |f(x) - g \circ \phi(x)| = 0$. On the other hand, by the construction of ψ^ε and Lemma 5.3.2, we have

$$\max_{y \in \widetilde{\mathcal{R}}} |g(y) - f \circ \psi^\varepsilon(y)| \leq \delta + \varepsilon.$$

We conclude that

$$\max\{\|f - g \circ \phi\|_\infty, \|f \circ \psi^\varepsilon - g\|_\infty\} \leq \delta + \varepsilon.$$

To bound the functional distortion distancebetween $\mathcal{R}$ and $\widetilde{\mathcal{R}}$ using Eqn 5.3, we now need to bound the term $D(\phi, \psi^\varepsilon)$ from Eqn 5.2. In particular, we wish to bound the distortion of distances for any pair of correspondences $(x_1, y_1), (x_2, y_2) \in G(\phi, \psi^\varepsilon)$; recall that $G(\phi, \psi^\varepsilon) = \{(x, \phi(x))\} \cup \{(\psi^\varepsilon(y), y)\}$ is the set of all correspondences induced by ϕ and ψ^ε. Assume that the two pairs (x_1, y_1) and (x_2, y_2) we have are of the form: $x_1 = \psi^\varepsilon(y_1)$ and $x_2 = \psi^\varepsilon(y_2)$. Below we will bound $|d_f(x_1, x_2) - d_g(y_1, y_2)|$.

Consider the ε-subdivision of $\widetilde{\mathcal{R}}$, and assume that y_1 falls in the arc $\tilde{\pi}(\tilde{v}_i, \tilde{v}_{i+1})$ of the subdivision, and y_2 falls in the arc $\tilde{\pi}(\tilde{v}_j, \tilde{v}_{j+1})$ in the subdivision. Both $\tilde{\pi}(\tilde{v}_i, \tilde{v}_{i+1})$ and $\tilde{\pi}(\tilde{v}_j, \tilde{v}_{j+1})$ are of height at most ε. Let v_i be the specific preimages of $\tilde{v}_i$ as chosen in the construction of ψ^ε; that is, $v_i = \psi^\varepsilon(\tilde{v}_i) \in \mu^{-1}(\tilde{v}_i)$. By the construction of ψ^ε, we have $x_1 \in \pi(v_i, v_{i+1})$ and $x_2 \in \pi(v_j, v_{j+1})$, where $\pi(v_i, v_{i+1})$ (resp. $\pi(v_j, v_{j+1})$) is the path connecting v_i to v_{i+1} as specified by Lemma 5.3.2. Now consider the optimal path $\tilde{\pi}(y_1, y_2)$ that gives rise to $d_g(y_1, y_2)$. Assume w.l.o.g. that the representation of $\tilde{\pi}(y_1, y_2)$ using arcs from the ε-subdivision of $\widetilde{\mathcal{R}}$ is as follows:

$$\tilde{\pi}(y_1, y_2) = \langle y_1, \tilde{v}_{i+1} = \tilde{v}_{I_0}, \tilde{v}_{I_1}, \ldots, \tilde{v}_{I_{s-1}}, \tilde{v}_j = \tilde{v}_{I_s}, y_2 \rangle,$$

where each $\tilde{v}_{I_a}$ is a vertex from the ε-subdivision of $\widetilde{\mathcal{R}}$. By Lemma 5.3.2, each arc $\tilde{\pi}(\tilde{v}_{I_a}, \tilde{v}_{I_{a+1}})$ gives rise to a path $\pi(v_{I_a}, v_{I_{a+1}})$ whose range is within δ-Hausdorff distance to range$(\tilde{\pi}(\tilde{v}_{I_a}, \tilde{v}_{I_{a+1}}))$. Let $\tilde{\pi}(\tilde{v}_{i+1}, \tilde{v}_j)$ denote the subpath $\langle \tilde{v}_{i+1} = \tilde{v}_{I_0}, \tilde{v}_{I_1}, \ldots, \tilde{v}_{I_{s-1}}, \tilde{v}_j = \tilde{v}_{I_s} \rangle$ of $\tilde{\pi}(y_1, y_2)$.

Concatenating all such $\pi(v_{I_a}, v_{I_{a+1}})$ together for $a \in [0, s-1]$, we obtain a path $\pi(v_{i+1}, v_j)$ in $\mathcal{R}$ whose range is within δ-Hausdorff distance from range$(\tilde{\pi}(\tilde{v}_{i+1}, \tilde{v}_j))$.

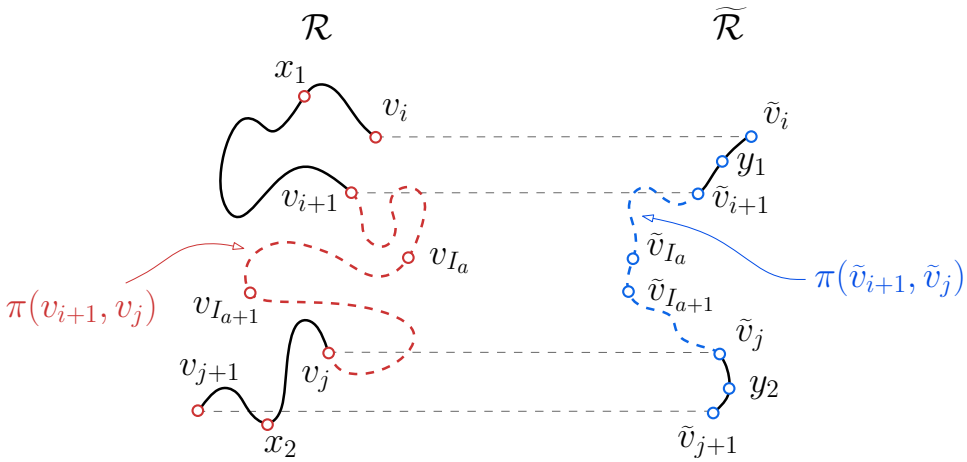


Figure 5.6: Illustration - I.

Furthermore, by Lemma 5.3.2, the range of the path $\pi(v_i, v_{i+1})$ is within δ-Hausdorff distance to range$(\tilde{\pi}(\tilde{v}_i, \tilde{v}_{i+1}))$. Since the monotone arc $\tilde{\pi}(\tilde{v}_i, \tilde{v}_{i+1}))$ has height at most ε, it then follows that $\pi(x_1, v_{i+1})$ (as a subpath of $\pi(v_i, v_{i+1})$) is within Hausdorff distance $\delta + \varepsilon$ to range$(\tilde{\pi}(y_1, \tilde{v}_{i+1}))$. A similar statement holds for the path $\pi(v_j, x_2)$. Putting everything together, we have that the path $\pi(x_1, v_{i+1}) \circ \pi(v_{i+1}, v_j) \circ \pi(v_j, x_2)$ from x_1 to x_2 satisfies that its range is within $(\delta+\varepsilon)$-Hausdorff distance from range$(\tilde{\pi}(y_1, y_2))$. Hence

$$d_f(x_1, x_2) \leq d_g(y_1, y_2) + 2\delta + 2\varepsilon.$$

Now let's consider the direction from $\mathcal{R}$ to $\widetilde{\mathcal{R}}$. Specifically, consider the optimal path $\pi^*(x_1, x_2)$ that gives rise to $d_f(x_1, x_2)$. It is mapped to a path $\tilde{\pi}^*(\tilde{y}_1, \tilde{y}_2) = \phi(\pi^*(x_1, x_2))$ connecting $\tilde{y}_1 = \phi(x_1)$ and $\tilde{y}_2 = \phi(x_2)$ in $\widetilde{\mathcal{R}}$ under the map $\phi = \mu : \mathcal{R} \to \widetilde{\mathcal{R}}$.

Furthermore, since ϕ is a quotient map that preserves function values, there always exists a pair of optimal path with range$(\tilde{\pi}^*(\tilde{y}_1, \tilde{y}_2)) =$ range$(\pi^*(x_1, x_2))$. Similarly,

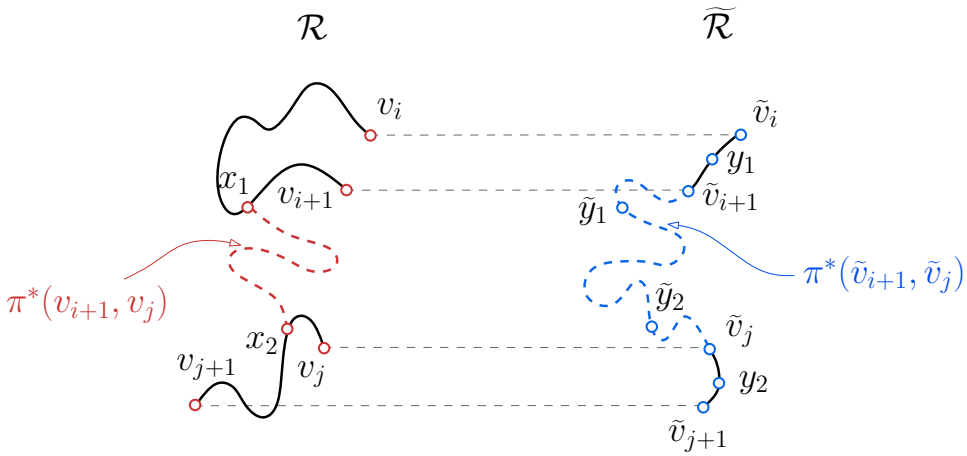


Figure 5.7: Illustration - II.

under ϕ, the path $\pi(x_1, v_i)$ (resp. $\pi(v_j, x_2)$ is mapped to a path $\pi'(\tilde{y}_1, \tilde{v}_i)$ (resp. $\pi'(\tilde{v}_j, \tilde{y}_2)$) of the same range. By Lemma 5.3.2, we have

$$\text{height}(\pi(x_1, v_i)) = \text{height}(\pi'(\tilde{y}_1, \tilde{v}_i)) \leq 2\delta + \varepsilon \tag{5.7}$$

$$\text{height}(\pi(x_1, v_{i+1})) \leq \text{height}(\pi'(\tilde{y}_1, \tilde{v}_{i+1})) + 2\delta\,. \tag{5.8}$$

A similar bound holds for the path $\pi(x_2, v_j)$ and $\pi(x_2, v_{j+1})$. Hence the path $\pi'(\tilde{v}_{i+1}, \tilde{y}_1) \circ \tilde{\pi}^*(\tilde{y}_1, \tilde{y}_2) \circ \pi'(\tilde{y}_2, \tilde{v}_j)$ is a path connecting $\tilde{v}_{i+1}$ to $\tilde{v}_j$ whose range is within $(2\delta + 2\varepsilon)$-Hausdorff distance to the range of $\pi^*(x_1, x_2)$. Since by the construction of the ε-subdivision, each arc $\tilde{\pi}(y_1, \tilde{v}_{i+1})$ and $\tilde{\pi}(y_2, \tilde{v}_j)$ is of height at most ε, we have that there is a path in $\widetilde{\mathcal{R}}$ connecting y_1 to y_2 whose range is within $(2\delta + 4\varepsilon)$-Hausdorff distance to the range of $\pi^*(x_1, x_2)$. In other words, $d_g(y_1, y_2) \leq d_f(x_1, x_2) + 4\delta + 8\varepsilon$. Putting everything together, we have

$$|d_f(x_1, x_2) - d_g(y_1, y_2)| \leq 4\delta + 8\varepsilon.$$

In other words,

$$D(\phi, \psi^\varepsilon) = \sup \frac{1}{2}|d_f(x_1, x_2) - d_g(y_1, y_2)| \leq 2\delta + 4\varepsilon.$$

Using an analogous argument, we obtain the same bound for the cases $y_1 = \phi(x_1), y_2 = \phi(x_2)$ and $y_1 = \phi(x_1), x_2 = \psi^\varepsilon(y_2)$. Putting everything together, we have that

$$d_{\phi,\psi^\varepsilon} := \max\{D(\phi, \psi^\varepsilon), \|f - g \circ \phi\|_\infty, \|f \circ \psi^\varepsilon - g\|_\infty\} \leq 2\delta + 4\varepsilon. \tag{5.9}$$

Let ε approach 0, we then have $d_{FD}(\mathcal{R}, \widetilde{\mathcal{R}}) \leq \lim_{\varepsilon \to 0} d_{\phi,\psi^\varepsilon} = 2\delta$. Combining this with Theorem 5.2.3 and Theorem 5.2.4, we thus obtain

$$\begin{cases} d_B(D_{\mathrm{g}_0}(\mathcal{R}_{\pm f}), D_{\mathrm{g}_0}(\mathcal{R}_{\pm g})) \leq d_{FD}(\mathcal{R}, \widetilde{\mathcal{R}}) = 2\delta, \\ d_B(ExD_{\mathrm{g}_1}(\mathcal{R}_f), ExD_{\mathrm{g}_1}(\mathcal{R}_g)) \leq 3d_{FD}(\mathcal{R}, \widetilde{\mathcal{R}}) = 6\delta. \end{cases} \tag{5.10}$$

5.3.3 An alternative distance bound for $\mathcal{R}$ and $\widetilde{\mathcal{R}}$

In this section, we bound functional Gromov-Hausdorff distance, $d_{fGH}(\mathcal{R}, \widetilde{\mathcal{R}})$, between $\mathcal{R}$ and $\widetilde{\mathcal{R}}$, and infer the bound of $d_{FD}(\mathcal{R}, \widetilde{\mathcal{R}})$ by Theorem 5.2.5. We do so by constructing continuous map $\phi : \mathcal{R} \to \widetilde{\mathcal{R}}$, which describes the simplification process implemented on $\mathcal{R}$ so that $\widetilde{\mathcal{R}}$ is obtained, and proving that the Gromov-Hausdorff distance between $\mathcal{R}$ and $\widetilde{\mathcal{R}}$ is bounded by 2δ.

Being analogous to Eqn 5.2, we define the Gromov Hausdorff distance between $\mathcal{R}$ and $\widetilde{\mathcal{R}}$ to be

$$d_{GH}(\mathcal{R}, \widetilde{\mathcal{R}}) = \inf_C \left(D(C(x, y))\right), \tag{5.11}$$

where C denotes the correspondence between the points $x \in \mathcal{R}$ and $y \in \widetilde{\mathcal{R}}$. Set $\widehat{C} = \{(x, \phi(x)) | x \in \mathcal{R}\}$ is indeed a such correspondence since ϕ is a subjective mapping

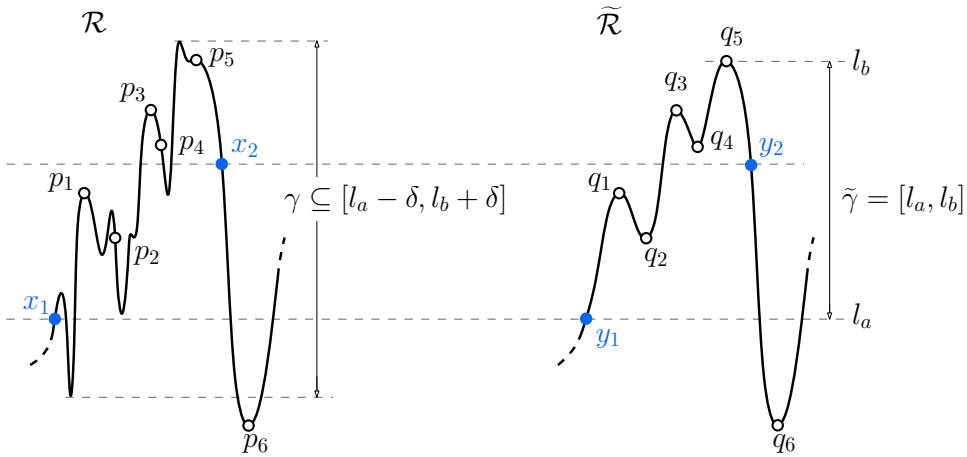


Figure 5.8: Right: An arc $(q_i, q_{i+1}) \in \widetilde{\mathcal{R}}$ denotes a monotonic path. Each of them has a correspondent path $(p_i, p_{i+1}) \in \mathcal{R}$, with $q_i = \phi(p_i)$. The path between y_1 and y_2 is the concatenation of a set of monotonic paths, i.e., $\tilde{\pi}(y_1, y_2) = \{(y_1, q_1,), \cdots, (q_5, y_2)\} \in \widetilde{\mathcal{R}}$ with range$(\tilde{\pi}(y_1, y_2)) = [l_a, l_b]$. Left: There exists a path $\pi(x_1, x_2)$ in $\mathcal{R}$, with x_1 and x_2 being the arbitrarily preimages of y_1 and y_2, respectively. The range of $\pi(x_1, x_2)$ can be bounded as $[l_a - \delta, l_b + \delta]$.

from $\mathcal{R}$ to $\widetilde{\mathcal{R}}$. Given any $\{(x_1, y_2), (x_2, y_2)\} \in \widehat{C}$ with $y_1 = \phi(x_1)$ and $y_2 = \phi(x_2)$, we show $|d_f(x_1, x_2) - d_g(y_1, y_2)| \leq 2\delta$, which can be rewritten as

$$-2\delta \leq d_f(x_1, x_2) - d_g(y_1, y_2) \leq 2\delta \tag{5.12}$$

To proof the left inequality in Eqn 5.12, we review the result shown in Observation 5.3.1 and obtain $d_g(y_1, y_2) - d_f(x_1, x_2) \leq 0$, which naturally infers the inequality

$$-2\delta \leq d_f(x_1, x_2) - d_g(y_1, y_2)$$

Now we show the right part of Eqn 5.12. For two points $y_1, y_2 \in \widetilde{\mathcal{R}}$, if they are connected, there exists a set of paths $\Pi = \{\pi^1, \pi^2, \cdots, \pi^m\} \in \widetilde{\mathcal{R}}$ with each π^j being a path connecting y_1, y_2. Let $\tilde{\pi}(y_1, y_2) \in \widetilde{\mathcal{R}}$ denote the path with the minimum height

in Π. W.l.o.g., assume $\tilde{\pi}(y_1, y_2) \in \widetilde{\mathcal{R}}$ is the concatenation of a set of monotonic paths

$$\tilde{\pi}(y_1, y_2) = \{\pi^*(y_1, q_s,), \pi^*(q_s, q_{s+1}), \cdots, \pi^*(q_i, q_{i+1}), \cdots, \pi^*(q_{e-1}, q_e), \pi^*(q_e, y_2)\} \in \widetilde{\mathcal{R}}$$

By Lemma 5.3.2, each $\tilde{\pi}^*(q_i, q_{i+1})$ gives rise to a path $\pi^*(p_i, p_{i+1}) \in \mathcal{R}$ with the relationship $\text{height}(\pi^*(p_i, p_{i+1})) \leq 2\delta + \text{height}(\tilde{\pi}^*(q_i, q_{i+1}))$. Concatenating all $\pi^*(p_i, p_{i+1})$ with $i = s, \cdots, e$, we obtain a path $\pi^*(p_s, p_e)$ with $\text{height}(\pi^*(p_s, p_e)) \leq \text{height}(\tilde{\pi}^*(q_s, q_e)) + 2\delta$. Since both $\tilde{\pi}^*(y_1, q_s)$ and $\tilde{\pi}^*(q_e, y_2)$ are also monotonic paths, now let $\tilde{\gamma} = \text{range}(\tilde{\pi}(y_1, y_2)) = [l_a, l_b]$ denote the range of path $\tilde{\pi}(y_1, y_2)$, and $\gamma = \text{range}(\pi(x_1, x_2))$ denote the range of path $\pi(x_1, x_2)$, we have $\gamma \subseteq [l_a - \delta, l_b + \delta]$. Namely, $\text{height}(\pi(x_1, x_2)) \leq \text{height}(\tilde{\pi}(y_1, y_2)) + 2\delta$. This proves the right part of Eqn 5.12, that is $d_f(x_1, x_2) - d_g(y_1, y_2) \leq 2\delta$.

By the definition of functional Gromov-Hausdorff distance shown in Eqn 5.5 and Eqn 5.11, we bound the functional Gromov-Hausdorff distance between reeb graph $\mathcal{R}$ and its simplified graph $\widetilde{\mathcal{R}}$ as

$$d_{fGH}(\mathcal{R}, \widetilde{\mathcal{R}}) \leq \frac{1}{2} |d_f(x_1, x_2) - d_g(y_1, y_2)| = \delta \tag{5.13}$$

Therefore, by Theorem 5.2.5, we have

$$d_{FD}(\mathcal{R}, \widetilde{\mathcal{R}}) \leq 3 d_{fGH}(\mathcal{R}, \widetilde{\mathcal{R}}) = 3\delta \tag{5.14}$$

Though the bounding shown above is looser than the one in Eqn 5.10, its proof is much shorter and more intuitive compared with the one in Section 5.3.2.

5.4 Conclusion Remark

We proposed a distance for Reeb graphs, under which the Reeb graph is stable with respect to the changes in the input function under the L_∞ norm. More importantly,

we show that this distance is bounded from below and thus more discriminative at differentiating scalar fields than the bottleneck distance between both 0th ordinary and 1st extended persistence diagrams. Similar to the use of the Gromov-Hausdorff distance for metric spaces, having the Reeb graph distance metric provides a formal language for describing and studying various properties of the Reeb graphs. By bounding the functional distortion distance between a Reeb graph and its simplified version, we can prove that the major features will be preserved under the Reeb graph simplification process.

Chapter 6: Final Remarks

In this dissertation work, we proposed a sequence of algorithms from feature aware sampling to 1D structure reconstruction, and demonstrated their effectiveness via various applications. These methods work reasonably well for both low dimensional data feature reconstruction and high dimensional data analysis and description.

A major part of this work is based on the concept of Reeb graph, which is an abstract graph providing descriptors of the hidden structure or objects in the input data. To handle the noisy input, we presented a natural way to simplify the Reeb graph by removing spurious graph features. Furthermore, we provide a theoretical study on the potential distortion that can be caused in such simplifications. By bounding the distortion distance, we prove that the important features are guaranteed to be preserved under the simplification, which addresses a key practical issue of the Reeb graph usage.

We now list some of the promising research directions.

Feature aware sampling for high-dimensional data. The feature aware sampling approach shown in Chapter 2 is mainly designed for the applications in lower dimensional space, e.g., image, 2D manifold or video. Though our feature aware metric is flexible enough to be extended to high dimensional space measurement,

the sampling algorithm like dart throwing or Voronoi diagram suffers the notorious *curse of dimensionality*. Namely, with the increasing of dimensionality, the time/space consumption increases dramatically. To this end, we believe that designing an appropriate and generic algorithm for sampling in high dimensional space could be an interesting direction to pursue.

Geometry embedding for 1D skeleton. Reeb graph is a topological concept and does not have a natural geometric embedding. However, in most applications (especially the applications in computer graphics and visualization), an appropriate geometry embedding is highly desired. Our current algorithm works reasonably well in detecting the junction area, however, it's still hard to decide the precise location for a junction point. It would be interesting to develop an automatic and robust method to determine the precise junction locations.

Another related problem under the same category is about the detection of concave corners which has been mentioned in Chapter 4. Those corner points, though sometimes are merely the regular points in a graph arc, impact the quality of the final reconstructions crucially. Our current Gaussian-weighted graph Laplacian approach has some inherent difficulties in detecting such concave points. Research work aiming at feature corner/junction points detecting and reconstruction would be another interesting direction to go.

Robust hidden graph reconstruction from data with statistical and outlier-type noise. Our current method is robust to Hausdorff noise, however, in practice, one often encounters data corrupted with statistical or outlier-type noise, which makes the problem become much more challenging. By this token, we believe designing an

algorithm for reconstructing hidden graphs from the data with more complicated noise would potentially be an interesting research topic which has various applications in data visualization and analysis.

Bibliography

[1] M. Aanjaneya, F. Chazal, D. Chen, M. Glisse, L. Guibas, and D. Morozov. Metric graph reconstruction from noisy data. In *Proc. 27th Sympos. Comput. Geom.*, 2011.

[2] Andrew Adams, Natasha Gelfand, Jennifer Dolson, and Marc Levoy. Gaussian KD-trees for fast high-dimensional filtering. *SIGGRAPH '09, ACM Trans. Graph.*, 28(3):21:1–21:12, July 2009.

[3] P. K. Agarwal, H. Edelsbrunner, J. Harer, and Y. Wang. Extreme elevation on a 2-manifold. *Discrete and Computational Geometry (DCG)*, 36(4):553–572, 2006.

[4] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. In *Proceedings of the sixteenth annual symposium on Computational geometry*, SCG '00, pages 213–222, New York, NY, USA, 2000. ACM.

[5] Michael Balzer, Thomas Schlömer, and Oliver Deussen. Capacity-constrained point distributions: a variant of Lloyd's method. *SIGGRAPH '09, ACM Trans. Graph.*, 28(3):86:1–86:8, July 2009.

[6] Francesco Banterle, Massimiliano Corsini, Paolo Cignoni, and Roberto Scopigno. A low-memory, straightforward and fast bilateral filter through subsampling in spatial domain. *Computer Graphics Forum*, 31(1):19–32, 2012.

[7] Ulrich Bauer, Xiaoyin Ge, and Yusu Wang. Measuring distance between reeb graphs. *CoRR*, abs/1307.2839, 2013.

[8] Ulrich Bauer, Xiaoyin Ge, and Yusu Wang. Measuring distance between reeb graphs. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, SOCG'14, pages 464:464–464:473, New York, NY, USA, 2014. ACM.

[9] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput.*, 15(6):1373–1396, June 2003.

[10] Mikhail Belkin, Qichao Que, Yusu Wang, and Xueyuan Zhou. Toward understanding complex spaces: Graph laplacians on manifolds with singularities and boundaries. In *COLT 2012 - The 25th Annual Conference on Learning Theory, June 25-27, 2012, Edinburgh, Scotland*, pages 36.1–36.26, 2012.

[11] Paul Bendich, Bei Wang, and Sayan Mukherjee. Local homology transfer and stratification learning. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1355–1370, 2012.

[12] Arindam Bhattacharya and Rephael Wenger. Constructing Isosurfaces with Sharp Edges and Corners using Cube Merging. *Computer Graphics Forum*, 2013.

[13] S. Biasotti, D. Giorgi, M. Spagnuolo, and B. Falcidieno. Reeb graphs for shape analysis and applications. *Theoretical Computer Science*, 392(13):5 – 22, 2008. Computational Algebraic Geometry and Applications.

[14] John Bowers, Rui Wang, Li-Yi Wei, and David Maletz. Parallel Poisson disk sampling with spectrum analysis on surfaces. *SIGGRAPH Asia '10, ACM Trans. Graph.*, 29(6):166:1–166:10, December 2010.

[15] Gunnar Carlsson, Vin de Silva, and Dmitriy Morozov. Zigzag persistent homology and real-valued functions. In *Proceedings of the Twenty-fifth Annual Symposium on Computational Geometry*, SCG '09, pages 247–256, New York, NY, USA, 2009. ACM.

[16] Jianghao Chang, Benoît Alain, and Victor Ostromoukhov. Structure-aware error diffusion. *SIGGRAPH Asia '09, ACM Trans. Graph.*, 28(5):162:1–162:8, December 2009.

[17] K. Chang and J. Grosh. A unified model for probabilistic principal surfaces. *IEEE Trans. Pattern Anal. Machine Intell.*, 24(1):59–64, 2002.

[18] F. Chazal, D. Cohen-Steiner, and A. Lieutier. A sampling theory for compact sets in Euclidean space. *Discrete Comput. Geom.*, 41(3):461–479, 2009.

[19] Frederic Chazal, Vin de Silva, Marc Glisse, and Steve Oudot. The structure and stability of persistence modules. 2013.

[20] Frédéric Chazal, David C. Steiner, Leonidas J. Guibas, Facundo Mémoli, and Steve Y. Oudot. Gromov-Hausdorff stable signatures for shapes using persistence. In *Proceedings of the Symposium on Geometry Processing*, SGP '09, pages 1393–1403, Aire-la-Ville, Switzerland, Switzerland, 2009. Eurographics Association.

[21] Frédéric Chazal and Jian Sun. Gromov-hausdorff approximation of filament structure using reeb-type graph. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, SOCG'14, pages 491:491–491:500, New York, NY, USA, 2014. ACM.

[22] Jiating Chen, Xiaoyin Ge, Li-Yi Wei, Bin Wang, Yusu Wang, Huamin Wang, Yun Fei, Kang-Lai Qian, Jun-Hai Yong, and Wenping Wang. Bilateral blue noise samples. In *SIGGRAPH Asia '13*, page submitted to, 2013.

[23] Jiawen Chen, Sylvain Paris, and Frédo Durand. Real-time edge-aware image processing with the bilateral grid. *SIGGRAPH '07, ACM Trans. Graph.*, 26(3), July 2007.

[24] Zhonggui Chen, Zhan Yuan, Yi-King Choi, Ligang Liu, and Wenping Wang. Variational blue noise sampling. *TVCG*, 2012.

[25] Siu-Wing Cheng, Tamal K. Dey, and Edgar A. Ramos. Delaunay refinement for piecewise smooth complexes. *Discrete Comput. Geom.*, 43:121–166, 2010.

[26] Kree Cole-McLaughlin, Herbert Edelsbrunner, John Harer, Vijay Natarajan, and Valerio Pascucci. Loops in reeb graphs of 2-manifolds. In *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, SCG '03, pages 344–350, New York, NY, USA, 2003. ACM.

[27] Robert L. Cook. Stochastic sampling in computer graphics. *ACM Trans. Graph.*, 5(1):51–72, 1986.

[28] Fernando de Goes, Katherine Breeden, Victor Ostromoukhov, and Mathieu Desbrun. Blue noise through optimal transport. *SIGGRAPH Asia '12, ACM Trans. Graph.*, 31(6):171:1–171:11, November 2012.

[29] T. K. Dey, X. Ge, Q. Que, I. Safa, L. Wang, and Y. Wang. Feature-preserving reconstruction of singular surfaces. *Comp. Graph. Forum*, 31(5):1787–1796, August 2012.

[30] T. K. Dey and Y. Wang. Reeb graphs: Approximation and persistence. In *Proc. 27th Sympos. Comput. Geom.*, 2011.

[31] Tamal K. Dey, Fengtao Fan, and Yusu Wang. Computing topological persistence for simplicial maps. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, SOCG'14, pages 345:345–345:354, New York, NY, USA, 2014. ACM.

[32] Tamal K. Dey and Lei Wang. Voronoi-based feature curves extraction for sampled singular surfaces. *Computers & Graphics*, 37(6):659–668, 2013.

[33] D Dong and T. J Mcavoy. Nonlinear principal component analysis based on principal curves and neural networks. *Computers & Chemical Engineering*, 20:65–78, 1996.

[34] Harish Doraiswamy and Vijay Natarajan. Efficient algorithms for computing reeb graphs. *Comput. Geom. Theory Appl.*, 42(6-7):606–616, August 2009.

[35] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal voronoi tessellations: Applications and algorithms. *SIAM Rev.*, 41(4):637–676, December 1999.

[36] T. Duchamp and W. Stuetzle. Extremal properties of principal curves in the plane. *The Annals of Statistics*, 24(4):1511–1520, 1996.

[37] Mohamed S. Ebeida, Scott A. Mitchell, Anjul Patney, Andrew A. Davidson, and John D. Owens. A Simple Algorithm for Maximal Poisson-Disk Sampling in High Dimensions. *Computer Graphics Forum*, 2012.

[38] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, FOCS '00, pages 454–, Washington, DC, USA, 2000. IEEE Computer Society.

[39] Herbert Edelsbrunner and John L. Harer. *Computational Topology*. American Mathematical Society, 2009.

[40] Raanan Fattal. Blue-noise point sampling using kernel density model. *SIGGRAPH '11, ACM Trans. Graph.*, 30(4):48:1–48:12, July 2011.

[41] Eduardo S. L. Gastal and Manuel M. Oliveira. Adaptive manifolds for real-time high-dimensional filtering. *SIGGRAPH '12, ACM Trans. Graph.*, 31(4):33:1–33:13, July 2012.

[42] Xiaoyin Ge, Issam Safa, Mikhail Belkin, and Yusu Wang. Data skeletonization via reeb graphs. In *NIPS*, pages 837–845, 2011.

[43] Xiaoyin Ge, Li-Yi Wei, Yusu Wang, and Huamin Wang. Bilateral blue noise sampling. Technical Report OSU-CISRC-4/13-TR11, The Ohio State University - Department of Computer Science, 2013. ftp://ftp.cse.ohio-state.edu/pub/tech-report/2013/TR11.pdf.

[44] Andrew S. Glassner, editor. *An Introduction to Ray Tracing*. Morgan Kaufmann Publishers, 1989.

[45] Gloria Haro, Gregory Randall, and Guillermo Sapiro. Translated poisson mixture model for stratification learning. *Int. J. Comput. Vision*, 80(3):358–374, December 2008.

[46] William Harvey, Yusu Wang, and Rephael Wenger. A randomized o(m log m) time algorithm for computing reeb graphs of arbitrary simplicial complexes. In *Proceedings of the 2010 annual symposium on Computational geometry*, SoCG '10, pages 267–276, New York, NY, USA, 2010. ACM.

[47] William Harvey, Yusu Wang, and Rephael Wenger. A randomized o(mlogm) time algorithm for computing reeb graphs of arbitrary simplicial complexes. In *Proceedings of the Twenty-sixth Annual Symposium on Computational Geometry*, SoCG '10, pages 267–276, New York, NY, USA, 2010. ACM.

[48] Trevor J. Hastie. *Principal curves and surfaces.* PhD thesis, stanford university, 1984.

[49] Trevor J. Hastie and Werner Stuetlze. Principal curves. *Journal of the American Statistical Association*, 84(406):502–516, 1989.

[50] Hui Huang, Shihao Wu, Minglun Gong, Daniel Cohen-Or, Uri Ascher, and Hao Zhang. Edge-aware point set resampling. *TOG*, 2012.

[51] Nigel J. Kalton and Mikhail I. Ostrovskii. Distances between Banach spaces. *Forum Mathematicum*, 11(1), March 2008.

[52] B. Kégl and A. Krzyżak. Piecewise linear skeletonization using principal curves. *IEEE Trans. Pattern Anal. Machine Intell.*, 24:59–74, January 2002.

[53] B. Kégl, A. Krzyzak, T. Linder, and K. Zeger. Learning and design of principal curves. *IEEE Trans. Pattern Anal. Machine Intell.*, 22:281–297, 2000.

[54] Ares Lagae and Philip Dutré. A procedural object distribution function. *ACM Trans. Graph.*, 24(4):1442–1461, 2005.

[55] Ares Lagae and Philip Dutré. A comparison of methods for generating Poisson disk distributions. *Computer Graphics Forum*, 21(1):114–129, 2008.

[56] Hongwei Li, Li-Yi Wei, Pedro V. Sander, and Chi-Wing Fu. Anisotropic blue noise sampling. *SIGGRAPH Asia '10, ACM Trans. Graph.*, 29(6):167:1–167:12, December 2010.

[57] Hua Li and David Mould. Structure-preserving stippling by priority-based error diffusion. In *GI '11*, pages 127–134, 2011.

[58] Yaron Lipman, Daniel Cohen-Or, David Levin, and Hillel Tal-Ezer. Parameterization-free projection for geometry reconstruction. *ACM Trans. Graph.*, 26(3), July 2007.

[59] Yang Liu, Wenping Wang, Bruno Lévy, Feng Sun, Dong-Ming Yan, Lin Lu, and Chenglei Yang. On centroidal voronoi tessellation—energy smoothness and fast computation. *ACM Trans. Graph.*, 28(4):101:1–101:17, September 2009.

[60] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[61] Chuanjiang Luo, Xiaoyin Ge, and Yusu Wang. Uniformization and density adaptation for point cloud data via graph laplacian. Technical Report OSU-CISRC-11/14-TR19, The Ohio State University - Department of Computer Science, 2014. ftp://ftp.cse.ohio-state.edu/pub/tech-report/2014/TR19.pdf.

[62] A. Witkin M. Kass and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1:321–331, 1988.

[63] Sanjit K. Mitra. *Digital Signal Processing*. McGraw-Hill Higher Education, 2005.

[64] Facundo Mmoli. Gromov-hausdorff distances in euclidean spaces. In *In Proc. Computer Vision and Pattern Recognition (CVPR)*, 2008.

[65] P. Niyogi, S. Smale, and S. Weinberger. Finding the homology of submanifolds with high confidence from random samples. *Discrete Comput. Geom.*, 39(1-3):419–441, 2008.

[66] U. Ozertem and D. Erdogmus. Locally defined principal curves and surfaces. *Journal of Machine Learning Research*, 12:1249–1286, 2011.

[67] A. Cengiz Öztireli, Marc Alexa, and Markus Gross. Spectral sampling of manifolds. *SIGGRAPH Asia '10, ACM Trans. Graph.*, 29(6):168:1–168:8, December 2010.

[68] Wai-Man Pang, Yingge Qu, Tien-Tsin Wong, Daniel Cohen-Or, and Pheng-Ann Heng. Structure-aware halftoning. *SIGGRAPH '08, ACM Trans. Graph.*, 27(3):89:1–89:8, August 2008.

[69] Salman Parsa. A deterministic o(m log m) time algorithm for the reeb graph. In *Proceedings of the twenty-eighth annual symposium on Computational geometry*, SoCG '12, pages 269–276, New York, NY, USA, 2012. ACM.

[70] Valerio Pascucci, Giorgio Scorzelli, Peer-Timo Bremer, and Ajith Mascarenhas. Robust on-line computation of reeb graphs: Simplicity and speed. *ACM Trans. Graph.*, 26(3), July 2007.

[71] Mark Pauly, Richard Keiser, and Markus Gross. Multi-scale feature extraction on point-sampled models. In *Proceedings of Eurographics*, 2003.

[72] Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., 2010.

[73] Sam Roweis and Lawrence Saul. Nonlinear dimensionality reduction by locally linear embedding. Science, pages 2323–2326, 2000.

[74] B. Scholkopf, A. Smola, and K.R. Muller. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Computation*, 10:1299–1319, 2000.

[75] Adrian Secord. Weighted Voronoi stippling. In *NPAR '02*, pages 37–43, 2002.

[76] Derek Stanford and Adrian E. Raftery. Finding curvilinear features in spatial point patterns: Principal curve clustering with noise. *IEEE Trans. Pattern Anal. Machine Intell.*, 22(6):601–609, 2000.

[77] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. Science, pages 2319–2323, 2000.

[78] Robert Tibshirani. Principal curves revisited. *Statistics and Computing*, 2:183–190, 1992.

[79] Robert Ulichney. *Digital halftoning*. MIT Press, Cambridge, MA, 1987.

[80] J. J. Verbeek, N. Vlassis, and B. Kröse. A k-segments algorithm for finding principal curves. *Pattern Recognition Letters*, 23(8):1009–1017, 2002.

[81] Li-Yi Wei. Parallel Poisson disk sampling. *SIGGRAPH '08, ACM Trans. Graph.*, 27(3):20:1–20:9, August 2008.

[82] Ben Weiss. Fast median and bilateral filtering. *SIGGRAPH '06, ACM Trans. Graph.*, 25(3):519–526, July 2006.

[83] Kenric B. White, David Cline, and Parris K. Egbert. Poisson disk point sets by hierarchical dart throwing. In *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, RT '07, pages 129–132, Washington, DC, USA, 2007. IEEE Computer Society.

[84] Dong-Ming Yan, Jianwei Guo, Xiaohong Jia, Xiaopeng Zhang, and Peter Wonka. Blue-noise remeshing with farthest point optimization. *Comput. Graph. Forum*, 33(5):167–176, 2014.

[85] Dong Ming Yan, Jianwei Guo, Bin Wang, Xiaopeng Zhang, and Peter Wonka. A survey of blue-noise sampling and its applications. *J. Comput. Sci. Technol.*, 30(3):439–452, 2015.

[86] Afra Zomorodian. Technical section: Fast construction of the vietoris-rips complex. *Comput. Graph.*, 34(3):263–271, June 2010.

[87] Afra Zomorodian and Gunnar Carlsson. Computing persistent homology. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, pages 347–356, New York, NY, USA, 2004. ACM.