Eurographics Symposium on Geometry Processing 2012
Eitan Grinspun and Niloy Mitra
(Guest Editors)

Volume 31 (2012), Number 5

Feature-Preserving Reconstruction of Singular Surfaces

T. K. Dey[1] and X. Ge[1] and Q. Que[1] and I. Safa[1] and L. Wang[1] and Y. Wang[1]

[1]Computer Science and Engineering Dept., The Ohio State University, U.S.A.

Abstract

Reconstructing a surface mesh from a set of discrete point samples is a fundamental problem in geometric modeling. It becomes challenging in presence of 'singularities' such as boundaries, sharp features, and non-manifolds. A few of the current research in reconstruction have addressed handling some of these singularities, but a unified approach to handle them all is missing. In this paper we allow the presence of various singularities by requiring that the sampled object is a collection of smooth surface patches with boundaries that can meet or intersect.

Our algorithm first identifies and reconstructs the features where singularities occur. Next, it reconstructs the surface patches containing these feature curves. The identification and reconstruction of feature curves are achieved by a novel combination of the Gaussian weighted graph Laplacian and the Reeb graphs. The global reconstruction is achieved by a method akin to the well known Cocone reconstruction, but with weighted Delaunay triangulation that allows protecting the feature samples with balls. We provide various experimental results to demonstrate the effectiveness of our feature-preserving singular surface reconstruction algorithm.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

Reconstructing a quality surface mesh from a set of discrete point samples is a fundamental problem in geometric modeling, with many applications in science and engineering. Numerous algorithms have been proposed for this problem [Dey06, GP07]. Most of the proposed algorithms focus on reconstructing smooth surfaces without boundaries. Recently, there have been growing interests in reconstructing surfaces while preserving sharp features such as non-smooth creases. However, in practice, data can be sampled from more general domains, including non-manifolds where multiple surface patches can meet or intersect. In this paper 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.

1.1. Our work

In this paper, we propose a simple yet effective reconstruction algorithm for singular surfaces that can handle all three singularities mentioned above in a unified framework. Our algorithm has two components (see Figure 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 [BQWZ12] and the Reeb graph [DW11, GSBW11]. Our approach provides a unified approach to handle all three types of singularities. 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 special handling. Our approach is robust to noise and can possibly be used in other applications involving point cloud processing, such as in stylish drawing [PKG03].

For the second step of feature-preserving singular surface reconstruction, we use a variant of the well known Cocone

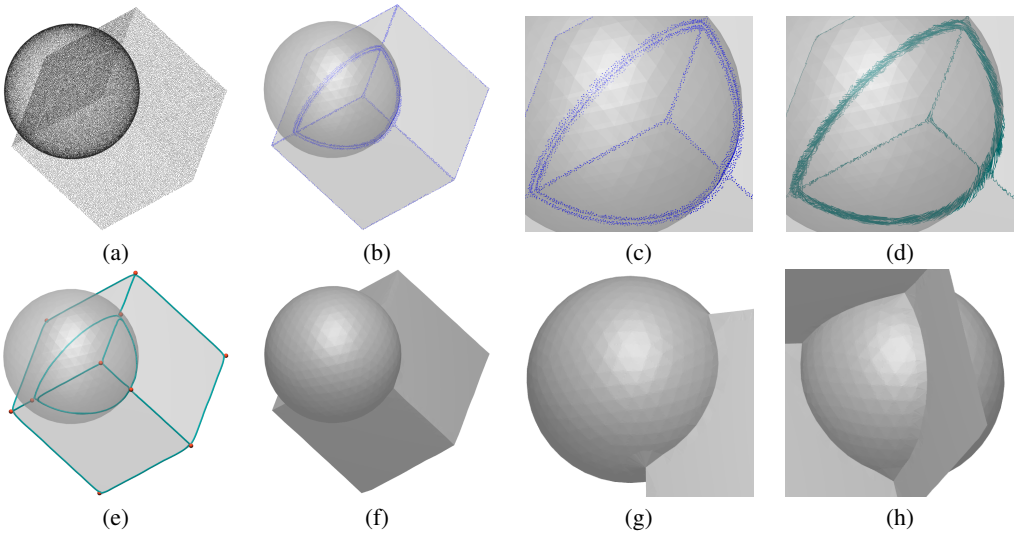


Figure 1: *Workflow of our algorithm. (a) Original point clouds. The hidden domain is a sphere intersecting a half-cube with only three faces (the three visible ones). (b) Feature points identified. (c) A zoom-in of feature points around intersections. (d) Coarse feature curves reconstructed. (e) Refined feature curves, with junction nodes and sharp corners marked. (f) Reconstructed singular surface. Two zoomed-in views of reconstructed model near intersection in (g) and (h).*

algorithm [ACDL02]. Inspired by the success of ball protection idea of [CDR10] in generating meshes from piecewise smooth complexes, we propose a weighted variant of the Cocone algorithm. 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, we put a protecting ball centering each sample point on 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. Only the unweighted points are allowed to choose the cocone triangles. This simple modification of the Cocone algorithm allows us to reconstruct singular surfaces quite effectively as our experiments show.

1.2. Related work

A multitude of algorithms have been proposed in the literature that use Voronoi diagram and Delaunay triangulation to determine the ultimate mesh triangles for reconstruction as we do in this paper. The work of Amenta and Bern [AB99] inspired a flurry of activities in this area [ACDL02, BC02, ACSTD07]; see the book and survey [Dey06, CG06] for details. However, none of these algorithms is concerned with the reconstruction of surfaces with singular features with the only exception of a recent work by Cazals and Cohen-Steiner [CCS12]. This algorithm reconstructs compacts provably but while preserving homotopy. Our goal here is to reconstruct singular surfaces while preserving the full topology and singular features. Unlike [CCS12], we cannot provide provable guarantee, but we demonstrate the effectiveness of our method by empirical results.

Recently, some approaches have been proposed that aim to reconstruct surfaces from point data while preserving sharp features. Many of them are based on implicit functions such as the moving least-square (MLS) framework and Poisson surfaces [ABCO*01, KBH06]. Earlier work employs anisotropic basis functions [DTS01, AA06] which show higher fidelity in reconstructing *smooth* surfaces with high curvature features. Recently, Reuter et al. [RJT*05] proposed a projection operator for surface reconstruction, based on the *enriched reproducing kernel particle approximation*, to respect sharp features. However, their algorithm requires points on sharp features to be given as input. Lipman et al. [LCOL07] defined a singularity indicator field (SIF) which measures, for each point, its potential to be on or near a singularity. They use this SIF to guide the local polynomial fitting in a data-dependent MLS framework to reconstruct surfaces while preserving high-frequency features. By using a continuous singularity indicator field, their algorithm eliminates the need for a cutoff threshold separating sharp features from non-sharp features. This allows them to reconstruct delicate singularities faithfully as well. However, their method cannot handle higher-order singularities in the hidden surface where multiple feature curves meet (such as a corner point). Later, Öztireli et al. [ÖGG09] proposed a robust implicit MLS based on local non-linear kernel regression. Avron et al. [ASGCO10] proposed an algorithm

to reconstruct sharp point set surfaces based on an L_1-sparse method. This surface reconstruction algorithm naturally preserves sharp features while at the same time, is also resilient to noises due to a global optimization procedure.

There is another line of recent work which explicitly extracts sharp features during the surface reconstruction process. In particular, based on the idea that sharp feature lines are located where the fitting error using a single piece of surface patch is large, Fleishman et al. [FCOS05] developed an iterative refitting algorithm, called robust MLS, to gradually grow piecewise smooth surface patches. Sharp feature lines are identified where discontinuity of surface occurs. This algorithm initiated a new line of attack for handling sharp features, and provided reasonable results even when the input points have missing data. Unfortunately, the algorithm is relatively expensive, and requires explicit processing for corners where multiple surface patches meet. Also the reconstructed feature lines may not be smooth. A later work in [IHOS07] uses a method based on this robust MLS algorithm to extract smooth feature curves. In [JWS08], Jenke et al. proposed another algorithm to classify and reconstruct each piecewise smooth surface patch using the so-called patch-graph, which helps to combine both local and global information to produce more stable classification of surface patches. By detecting boundary points separately and explicitly, this algorithm can also handle surfaces with boundaries.

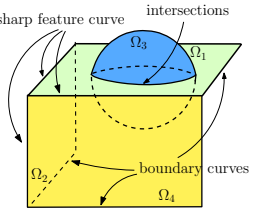
Another related work [MW11] can handle non-orientable and self-intersecting surfaces. Specifically, non-manifold regions are identified by local fitting and a triangulation is then constructed by an incremental (non Delaunay-based) algorithm. This algorithm does not preserve sharp features.

Our algorithm is most closely related to the work in [SYM10] in terms of the high level two-step framework: The algorithm in [SYM10] first identifies feature points based on the covariance matrices of Voronoi cells of data points as developed in [MOG09]. It then fits polylines through the feature points as feature curves, and reconstructs surfaces while preserving these feature curves based on a combination of the meshing algorithm of [BO05] and the protection-balls idea of [CDR10]. This algorithm outputs explicit sharp feature curves, in addition to the feature-preserving surface reconstruction. The algorithm shows robust reconstruction results under noise and sparse sampling.

To date, all the previous sharp-feature preserving reconstruction algorithms recover a manifold surface (mostly closed surfaces, with few exceptions [FCOS05, JWS08, MW11]). Our algorithm handles not only closed surfaces with sharp crease lines, but also non-manifold surfaces with boundaries, within a single framework. Furthermore, by using the Reeb graph to recover the feature graphs (i.e, feature curves with their connections), we can recover junction nodes easily and reliably, while previous approaches require special handling to discover them. Points along the extracted feature curves are fed to the subsequent reconstruction step, which employs a weighted version of Cocone algorithm to reliably reconstruct non-manifold singular surfaces with sharp features.

2. Algorithm Overview

Suppose we have a collection of smooth 2-manifolds with boundary $\{\Omega_1, \ldots, \Omega_k\}$ isometrically embedded in $I\!R^3$. We call each Ω_i a *surface patch*. These surface patches may intersect each other in their interior, giving 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 the right figure and Figure 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 P sampled from the singular surface Ω, which is the union of Ω_is, our goal is to reconstruct Ω while preserving various types of feature curves.

Our approach consists of two steps. In the first step, we use a combination of Gaussian-weighted graph Laplacian with the Reeb graph to identify and reconstruct a set of feature curves. These curves are sampled to generate a point set F. We next apply a weighted Cocone algorithm to reconstruct the singular surface, taking the point cloud $P \cup F$ as input. The algorithm is outlined below. The details of the two components are discussed in Section 4 and 5, respectively.

Step 1 (Feature Curves Identification and Reconstruction):

(1.a) Identify feature points via graph Laplacian
(1.b) Reconstruct feature curves via Reeb graphs
(1.c) Refine feature curves and generate a point sample F on them.

Step 2 (Feature-aware singular surface reconstruction):

(2.a) Put a protecting ball centering each sample point in F where the weights are determined by a parameter used for feature curve identification
(2.b) Delete points in P inside each protecting ball and let P' be the resulting set. Compute a weighted Delaunay triangulation of the point set $P' \cup F$
(2.c) Compute a cocone surface for each patch using the Cocone algorithm with the modification that only unweighted points in $P' \cup F$ choose cocone triangles.

3. Gaussian-weighted Graph Laplacian

The feature points identification algorithm uses the widely used Gaussian-weighted graph Laplace operator which we describe now. Given a set of points $P = \{p_1, \ldots, p_n\} \subset I\!R^3$,

the (un-normalized) Gaussian-weighted graph Laplace operator L_t is an $n \times n$ matrix where

$$\mathsf{L}_t[i][j] = \begin{cases} -\frac{1}{nt^2} e^{-\frac{\|p_i - p_j\|^2}{t}}, & \text{if } i \neq j; \\ \frac{1}{nt^2} \sum_{k=1, k\neq i}^{n} e^{-\frac{\|p_k - p_i\|^2}{t}}, & \text{if } i == j. \end{cases} \quad (1)$$

For any fixed function $f : P \to I\!R$ and a point $p_i \in P$, it is easy to verify that L_t applied to this function f is:

$$\mathsf{L}_t f(p_i) = \frac{1}{nt^2} \sum_{j=1}^{n} e^{-\frac{\|p_i - p_j\|^2}{t}} [f(p_i) - f(p_j)].$$

It was shown in [BN03] that if points in 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, $\mathsf{L}_t f(p)$ converges to $\Delta f(p)$ where Δ is the Laplace-Beltrami operator for M. The connection to Δ_{M} is made via the so-called functional Laplacian [BN03]:

$$\mathcal{L}_t f(p) = \frac{1}{t^{\frac{d}{2}+1}} \int_{\mathsf{M}} e^{-\frac{\|x-p\|^2}{t}} (f(p) - f(x)) dx.$$

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

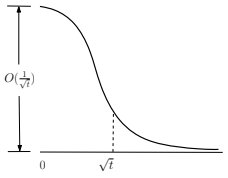
$$\mathcal{L}_t f(p) = \Delta f(p) + o(1). \quad (2)$$

Now suppose that the underlying domain where data are sampled from is not a manifold, but a singular d-manifold Ω. Interestingly, it turns out that the functional Laplacian $\mathcal{L}_t$ behaves differently around various singularities (i.e, feature curves in 2D) from around a regular point (see [BQWZ12] for details). In particular, for a point p lying on a boundary-feature curve of some Ω_i, we have that

$$\mathcal{L}_t f(p) = \frac{1}{\sqrt{t}} \frac{\pi^{\frac{1}{2}}}{2} \partial_{\vec{\mathbf{n}}} f(p) + o(\frac{1}{\sqrt{t}}), \quad (3)$$

where $\vec{\mathbf{n}}$ is the unit outward normal to the boundary-feature curve at p (that is, $\vec{\mathbf{n}}$ is in the tangent space of Ω_i at p and normal to the boundary-feature curve); and $\partial_{\vec{\mathbf{n}}} f(p)$ is the directional derivative of f in direction $\vec{\mathbf{n}}$. We remark that the term $\Delta f(p)$ as well as terms depending on intrinsic curvatures at p are now hidden in the lower-order $o(\frac{1}{\sqrt{t}})$ term which does not dominate for small t.

Comparing with Eqn (2), we see that while $\mathcal{L}_t f(p)$ reflects the manifold Laplacian (a second-order differential quantity) at a regular point, it reflects a first-order partial derivative at a boundary point. More importantly, the scale dependence on t is different: $O(1)$ for a regular point versus $O(\frac{1}{\sqrt{t}})$ for a singular point. For small t (which is usually the case in practice), $\frac{1}{\sqrt{t}}$ is large, implying that $\mathcal{L}_t f(p)$ is significantly larger at a boundary point than at a regular point (assuming $|\partial_{\vec{\mathbf{n}}} f(p)|$ 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 it follows a Gaussian distribution centered at a boundary point with variance t; thus this effect wears off rapidly. See the adjacent figure for an illustration of the first term of Eqn (3) (the y-axis in the figure) as the distance of point x to the boundary-feature curve (the x-axis in the figure) increases.

The cases for points on or around the other two types of features curves are similar, although more complicated. We briefly summarize the behavior of $\mathcal{L}_t f$ around those feature curves in Appendix A. As mentioned earlier, L_t is simply a discretization of $\mathcal{L}_t$ and thus shares the same behavior as $\mathcal{L}_t$. Our algorithm leverages the different scaling behavior of L_t around feature curves to help identify feature points.

Non-uniform sampling. Suppose the input points P are sampled from a non-uniform density function $\rho : \Omega \to I\!R$ which is smooth on each surface patch Ω_i. It turns out that under mild assumptions on ρ, the different scaling behavior of L_t still holds. In particular, the $O(\frac{1}{\sqrt{t}})$ term will now simply be multiplied by a factor of $\rho(x)$[†]. Hence our feature identification algorithm is reasonably robust against non-uniform sampling of the input domain.

4. Feature Curves Identification and Reconstruction

4.1. Potential feature points identification

Let the input point cloud be $P = \{p_1, \ldots, p_n\}$. We apply the graph Laplacian L_t to the coordinate functions $\mathrm{X}, \mathrm{Y}, \mathrm{Z} : I\!R^3 \to I\!R$, where $\mathrm{X}(p) = p.\mathrm{x}$, $\mathrm{Y}(p) = p.\mathrm{y}$ and $\mathrm{Z}(p) = p.\mathrm{z}$. Let $\overrightarrow{\mathbf{P}} = [\mathrm{X}\ \mathrm{Y}\ \mathrm{Z}]$ denote the coordinate functions restricted to the input points P, which is a $3 \times n$ matrix and can be considered as n 3-dimensional vectors. Applying L_t to $\overrightarrow{\mathbf{P}}$ gives rise to another list of 3-dimensional vectors $\overrightarrow{\mathbf{V}} = \mathsf{L}_t \overrightarrow{\mathbf{P}}$, where the ith row $\vec{\mathbf{v}}_i := \overrightarrow{\mathbf{V}}[i]$ is a 3-dimensional vector associated with point $p_i \in P$. Let V_t be the n-dimensional vector (function) where $V_t[i] := \|\vec{\mathbf{v}}_i\|$ is the norm of vector $\vec{\mathbf{v}}_i$.

It is known (see e.g, [DMSB99]) that $\Delta \overrightarrow{\mathbf{P}}(p) = H_p \cdot \vec{\mathbf{n}}_p$ for a regular point p from a smooth surface, where H_p is the mean-curvature at p and $\vec{\mathbf{n}}_p$ is the unit surface normal at p. Hence

$$\vec{\mathbf{v}}_i = \mathsf{L}_t \overrightarrow{\mathbf{P}}(p_i) \approx H_{p_i} \cdot \vec{\mathbf{n}}_{p_i}, \quad \text{at a regular point } p_i.$$

However, if p_i is on or near the three-types of feature curves we consider, $\vec{\mathbf{v}}_i$ reflects fundamentally different information.

[†] We note that this is not true at a regular point: for points sampled from a density function ρ, $\mathcal{L}_t f(x)$ converges to $-\rho(x)\Delta_{\rho^2} f(x)$ at a regular point, whereas the weighted Laplacian $\Delta_{\rho^2} f(x)$ equals $\frac{1}{\rho^2} \mathrm{div}[\rho^2 \mathrm{grad} f(x)]$; see e.g, [Gri06] for details.

Specifically, consider a point p on the boundary of a surface patch Ω_j. Let $\vec{\mathbf{n}} = [\vec{\mathbf{n}}.x\ \vec{\mathbf{n}}.y\ \vec{\mathbf{n}}.z]^T$ denote the unit outward normal vector to the boundary curve at p (note, $\vec{\mathbf{n}}$ is not the surface normal at p, but the unit vector in the tangent plane at p normal to the boundary curve). By Eqn (3), setting $C = \frac{\pi^{1/2}}{2}$ we have that

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

where $[1,0,0]^T$ is the unit vector in the x-direction, and $\langle , \rangle$ stands for the inner product. By using $\approx$, we omit lower order terms $o(\frac{1}{\sqrt{t}})$. Similarly, we have that $\mathcal{L}_t \mathrm{Y}(p) \approx \frac{C}{\sqrt{t}} \vec{\mathbf{n}}.y$ and $\mathcal{L}_t \mathrm{Z}(p) \approx \frac{C}{\sqrt{t}} \vec{\mathbf{n}}.z$. Putting them together, we have that

$$\mathsf{L}_t \overrightarrow{\mathbf{P}}(p) \approx \mathcal{L}_t \overrightarrow{\mathbf{P}}(p) \approx \frac{C}{\sqrt{t}} [\vec{\mathbf{n}}.x, \vec{\mathbf{n}}.y, \vec{\mathbf{n}}.z]^T = \frac{1}{\sqrt{t}} \cdot \frac{\pi^{1/2}}{2} \cdot \vec{\mathbf{n}}.$$

Note that $\mathsf{L}_t \overrightarrow{\mathbf{P}}(p)$ is independent of the choice of the coordinate system. The magnitude of $\mathsf{L}_t \overrightarrow{\mathbf{P}}(p)$ is simply $\frac{\pi^{1/2}}{2\sqrt{t}}$. As a point moves away from the boundary, the magnitude of $\mathsf{L}_t \overrightarrow{\mathbf{P}}(p)$ decreases rapidly from $\frac{\pi^{1/2}}{2\sqrt{t}}$ to the mean curvature (i.e, the case for regular points), following a Gaussian distribution with variance t. The direction of $\mathsf{L}_t \overrightarrow{\mathbf{P}}(p)$ also changes from the outward normal to the boundary curve to the surface normal for a regular point. Points on and around other types of feature curves share similar behavior.

In other words, $V_t[i]$ is the mean curvature (and thus of order $O(1)$) for a regular point p_i. But $V_t[i]$ is of order $\Theta(\frac{1}{\sqrt{t}})$ for a point on and near feature curves. By "near", we mean points within $O(\sqrt{t})$ distance away from feature curves, where constant hidden in the big-O notation depends on the specific singularity we have. Hence our algorithm identifies potential feature curve points as those whose V_t value is above a given threshold τ.

Examples. In Figure 1 (b) and (c), we 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 [BQWZ12], points *on* intersection-feature curves have low $\mathsf{L}_t \overrightarrow{\mathbf{P}}$ magnitude, but points *around* them have high values and are captured as feature points. Hence around an intersection-feature curve, there are two small bands of feature points (see Figure 1 (c)). We will see later that our feature curve reconstruction algorithm is able to close the gap between these two bands.

Implementation. Given a parameter t, we first compute the proximity graph from the input where every point $p \in P$ is connected to all neighbors within $5\sqrt{t}$ distance to p. Let $G = (P, E)$ denote the resulting proximity graph. We then build the matrix L_t as introduced in Eqn (1) but only restricted to edges in G: that is, if there is no edge between p_i and p_j, then $\mathsf{L}_t[i][j] = 0$. This results into a sparse matrix L_t. In our current implementation, we use a kd-tree structure to help compute the proximity graph G and hence L_t.

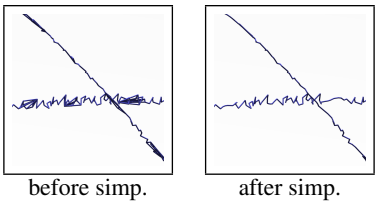
4.2. Coarse feature curves reconstruction

Once we compute a set of potential feature points Q from input point clouds P, we aim to produce feature curves (graphs) from them. For the subsequent surface reconstruction step we do not need the feature curves per se. However, we still need to generate sample points on the feature curves and identify the branching points which necessitates reconstructing them from Q. 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 (such as smooth feature lines aligned with boundaries or sharp features). To this end, we further refine the feature graph.

The feature points we obtain are around the feature curves that we wish to reconstruct. The work in [DW11] suggests that the hidden graph structure can be extracted from a certain Rips complex from these sampled points. Hence we first build a simplicial complex K from Q to connect discrete points, where K is Rips complex of Q using 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 $p, q, u \in$ Q are in K, we also add the triangle pqu to K.

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 "hugs" the feature curves (including closing the gap around intersection-feature curves as seen in Figure 1 (c) and (d)). Next, the algorithm of [GSBW11] computes the Reeb graph of some specific function defined on K to capture the skeleton of the the underlying space of K. The output is a graph where each branch is a polygonal curve with vertices being input feature points in Q. The branching nodes where multiple feature curves meet are obtained naturally as nodes in the Reeb graph. Previously, special handling is often required to detect and compute such nodes (see e.g, [IHOS07, SYM10]).


before simp. after simp.

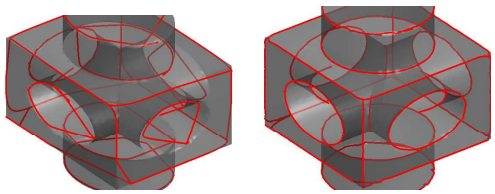
Furthermore, we can easily simplify the resulting feature graphs by the simplification of the Reeb graphs to remove noise and less important loops or branches. See the above figure for an example where several small spurious loops in the Reeb graph are removed.

4.3. 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 are not smooth, and may not be aligned with real features. In this step, we wish to improve the quality of the geometry of feature graphs. There are two components involved.

4.3.1. 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" of feature points involved. This causes shrinkage for boundary curves and sharp feature curves as they are smoothed; see Figure 2 (a). To align with real features while smoothing, we use an active-contour based method to deform the feature curve, similar to [PKG03].



(a) (b)

Figure 2: *(a) Laplacian smoothing causes feature curves shrink inside the model. (b) Our active contour approach alleviates this problem.*

In particular, we design the following energy function:

$$E_{\text{snake}} = \sum_{k=1}^{n} [E_{int}(q_k) + E_{ext}(q_k)],$$

where q_k is the k-th vertex of current piece of feature curve we are smoothing. The first term $E_{int}(q_k)$ aims to ensure the smoothness of the feature curve at q_k, and is the same as the one introduced in the original active-contour work [KWT88] based on approximated derivatives using finite differences. The second term E_{ext} aims to align curves with real features, and is defined as

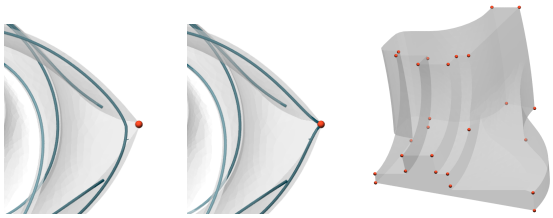
$$E_{ext}(q_k) = -\sum_{i=1}^{n} e^{-\frac{\|p_i - q_k\|^2}{\sigma^2}} V_t(p_i)$$

where σ is taken as $2\sqrt{t}$ in experiments. Note that in practice, we only consider points within $2\sigma = 4\sqrt{t}$ distance from q_k to compute $E_{ext}(q_k)$.

Intuitively, to minimize $E_{ext}(q_k)$, we want to maximize the sum of $e^{-\frac{\|p_i - q_k\|^2}{\sigma^2}} V_t(p_i)$ for points p_is with high V_t values. Hence we want to relocate q_k close to points with high V_t values to maximize this sum. For boundary and sharp feature curves, points with high V_t value lie along these singularities, and this energy term pushes q_k towards them. For intersection-type of singularity, points with high V_t values are around the intersection lines but not on them (recall Figure 1 (c) and Figure 8 (c) in Appendix A). However, due to symmetry, it is still beneficial to put q_k in the middle of these band of high V_t values, which is on the intersection curves. 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 a single simple framework. Since we have the explicit form for $E_{ext}(q)$, we can compute its gradient directly in closed form, which helps to reduce the computational cost.

4.3.2. Locations of sharp feature corners

To use the active contour approach described above for refining each branch in the feature graph, we wish to fix their endpoints, which correspond to the graph nodes. They can represent either a node where multiple feature curve pieces meet (graph nodes of degree 3 or more) or a tip of feature curves (degree-1 graph nodes). We also want to preserve sharp corners within a single feature curve: see Figure 3 (a) and (b); such degree-2 corners are not available in the Reeb graph and have to be identified separately.



(a) (b) (c)

Figure 3: *(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.*

To identify degree-2 sharp corners, we simply compute the local maxima of the function V_t (the magnitude of $\mathsf{L}_t \overrightarrow{\mathbf{P}}$) 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 small number of points available around such corners. So we also compute the normalized version of the weighted graph Laplacian, $\overline{\mathsf{L}}_t$, and also take the local maxima of the magnitude of $\overline{\mathsf{L}}_t \overrightarrow{\mathbf{P}}$. This does not incur any extra time complexity: when our algorithm computes the standard graph Laplacian $\mathsf{L}_t \overrightarrow{\mathbf{P}}$, we also compute the normalized version and store both values at each point.

Corners where 3 or more feature curves meet appear automatically as nodes in the Reeb graph. For each Reeb graph node p_i, let $NN(p_i)$ denote its neighbors within the Rips

complex K. Our goal is to find a good location for p_i to align with the sharp geometric corner (0-dimensional singularities) of the hidden domain. Simply taking the centroid of points in $NN(p_i)$ does not serve as a good choice as it tends to push p_i off the domain and away from the sharp corners. So we again take the point with largest V_t value in $NN(p_i)$ as the new position for the corner node p_i. See Figure 3 for an example of the degree-3 nodes that our algorithm computes.

5. Feature-Aware Singular Surface Reconstruction

5.1. Weighted-Cocone reconstruction

The active contour algorithm described in the previous section deforms feature curves. Let F denote the sample points on the deformed feature curves, with sharp nodes (of degree-2 or more) marked. We first take a sparse subset of feature points in F so that no two of them are within $0.8r$ distance to each other, where r is taken as the radius of protection balls around these points. Let F still denote the resulting sparsified feature curve samples. The radii r of the balls are taken as $\sqrt{t}$ to reflect the fact that anomalies due to the presence of features are assumed to have an influence radius of roughly $\sqrt{t}$ while identifying them. For the corner points, we make an exception that their protecting balls are made larger by taking their radii $3\sqrt{t}$. All points from P that are contained within the protection balls are removed. Let the remaining set of points be still denoted as P. The points in P do not have any ball, or equivalently, a ball of radius zero.

A sample point p with a ball of radius $r_p \geq 0$ centering it is turned into a weighted point $\hat{p} = (p, r_p)$. The distance between two weighted points $\hat{p}, \hat{q}$ is measured as the weighted distance $\|p-q\|^2 - r_p^2 - r_q^2$. Notice that any or both of r_p and r_q could be zero. With this weighted distance, one can define the weighted Voronoi diagram whose dual is the Weighted Delaunay triangulation of $(P \cup F)$. We apply the Cocone algorithm on this weighted Delaunay triangulation.

The original Cocone algorithm for a unweighted set of points P works as follows. For each point $p \in P$, the algorithm computes a pole vector from the Voronoi cell of p which is known to approximate the normal vector at p on the sampled surface. Then, a double cone with apex at p and an angle of $\pi/8$ with the pole vector is considered. All Voronoi edges in the Voronoi cell of p intersecting the complement of this double cone, also called the co-cone of p, are determined and their dual Delaunay triangles are chosen as the first approximation of the sampled surface. A subsequent manifold extraction step further prunes this initial approximation to compute the output surface mesh. We follow the same procedure to extract the mesh approximating the sampled singular surface with the modification that the entire computation is performed on the weighted Delaunay triangulation that accounts for the weighted points on the feature curves. The weighted points do not participate in filtering the triangles; that is, the triangles dual to the Voronoi edges that are intersected by the co-cones of the un-weighted points are selected. The rationale behind this choice is that normal estimation at and near the feature curves is generally unreliable. The balls around these points act as a guard so that un-weighted points are sufficiently away from these feature curves. The triangles adjoining weighted points in the final reconstruction are chosen by adjacent un-weighted points. The assumption is that, because of the dense sampling, each triangle adjoining a weighted point in the final reconstruction is chosen by an un-weighted point. Our experiments show that this assumption is not unreasonable.

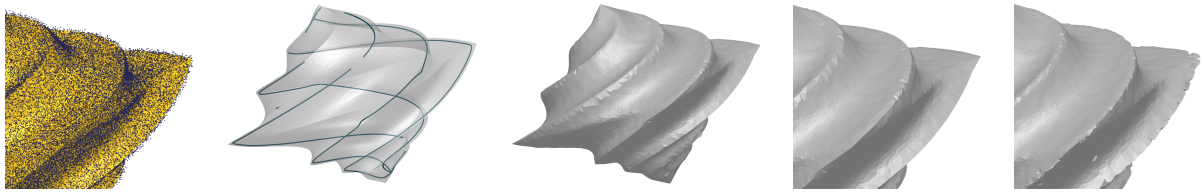
5.2. Reconstruction from noisy data

Our feature curve reconstruction step is reasonably robust against noise. However, the input for weighted Cocone algorithm should be noise-free. When input data contains noise, we perform the following to reconstruct our singular surfaces. First, we extract feature curves using the method described in Section 4. Next, we perform a small number of iterations of the version of the mean-shift algorithm proposed in [WCPn10] to denoise the input data. Note that similar to most denoising algorithms, mean-shift algorithm tends to smooth out sharp features: this is one reason why we extract feature curves before the denoising step. In this way, even though sharp features are smoothed after mean-shift, they are still recorded as feature curves, and thus are reconstructed by our weighted Cocone algorithm which operates on smoothed data points. See Figure 4, where we obtain a noisy sample of OctaFlower by perturbing each point randomly within 1% of the diagonal of the bounding box. In (c) we show the reconstructed surface by our algorithm, and in (d) we zoom-in to show the detail of one part. In (e) we show the surface reconstructed by the original Cocone algorithm from the smoothed (denoised) data set, where we can see that sharp features are not well-reconstructed due to missing triangles and smoothing. However, by preserving the feature curves (shown in (b)), which we compute before denoising, we can reconstruct these sharp features. More examples on reconstruction from noisy data are shown in Figure 6.

6. Experimental Results

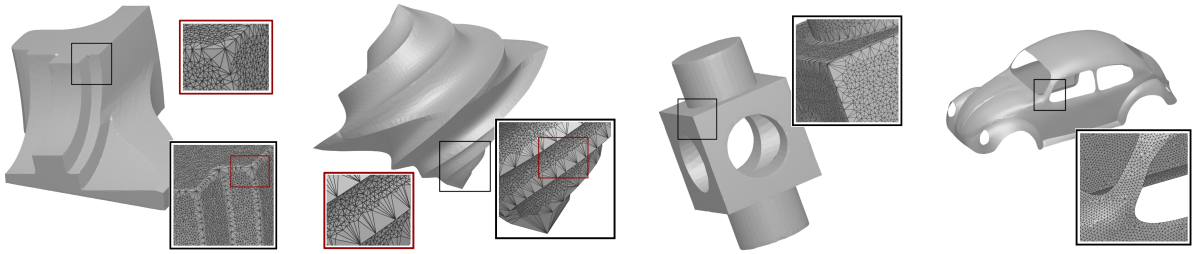
In this section, we first provide several examples to show that our algorithm can reconstruct singular manifold while preserving various singularities faithfully. See Figure 5. We also show more examples of reconstruction from noisy data in Figure 6, including the reconstruction of the non-manifold SphereCube model. In both cases, each point is perturbed randomly within 1% of the size of the diagonal of the bounding box of the model.

To exhibit the behavior of our algorithm with respect to the sparsification of the input, we show in Figure 7 the reconstruction of the block model from point cloud of different sizes. We start with an input of 74K points, and down-sample



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

Figure 4: *(a) Noisy points and (b) feature curves reconstructed from noisy point cloud, both overlayed on top of the clean surface model for visualization purpose. (c) and (d): Reconstructed surface by our algorithm. (e) Reconstruction using the original Cocone algorithm from the denoised (smoothed) points.*



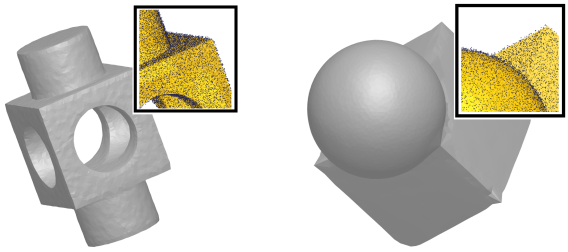
(a) (b) (c) (d)

Figure 5: *Reconstruction for various models.*

it to 47K, 21K and 8.5K points. Note that small mistakes (including dents along sharp feature lines and small missing triangles near the feature curves) start to appear in the model of size $21K$ which worsens to more global mistakes (large triangles connecting different parts of the model) as the input sparsifies to 8.5K points. We observed though if the detected feature curves were sampled with more density in 8.5K point model, the reconstruction would not incur global mistakes.

Parameters. The main parameter involved in our algorithm is t, and we set $\sqrt{t}$ as three times the average distance from a point to its 5th nearest neighbor. The default values of other parameters depend on $\sqrt{t}$, except for τ, the threshold to decide potential feature points, which we choose as three times the average V_t of all points. We use $\sqrt{t}$ as the parameter to build the Rips complex. We use $\sqrt{t}$ as the radius of protection ball for feature points in the singular manifold reconstruction step: we observe that corner points in the reconstruction step requires a bigger protection ball radius, so that no triangles are formed between points from different feature lines. Hence we choose $3\sqrt{t}$ to be the protection ball size for corner points. For the noisy points data, we have to increase the size of t so that $\sqrt{t}$ is larger than the noise level.

Timing. We have implemented a proto-type of our algorithm. The code is not yet optimized. Our current implementation uses a kd-tree to speed up computation of proximity graph. In Table 1 we show the timing for our experiments.



(a) (b)

Figure 6: *Reconstruction from noisy models for (a) block model and (b) SphereCube model.*

The experiments were carried out on a mac-laptop with Intel Core i7 2.2 Ghz, and 4 GB RAM. The timing is broken into 5 stages: kd-tree construction; computation of graph Laplacian and feature points detection; feature curve construction; feature curve refinement; surface reconstruction.

Models	Stg 1	Stg 2	Stg 3	Stg 4	Stg 5
SphCube (65K)	3.3	6.46	0.04	15.01	8.8
Fandisk(114K)	9.9	11.2	0.49	40.00	25.4
Block(47K)	1.8	1.93	0.07	6.30	7.1
Flower(107K)	8.76	13.8	0.13	23.01	13.4
Beetle(63K)	3.13	3.28	0.36	10.8	17.42

Table 1: *Timing (in seconds) for each stage of our algorithm.*

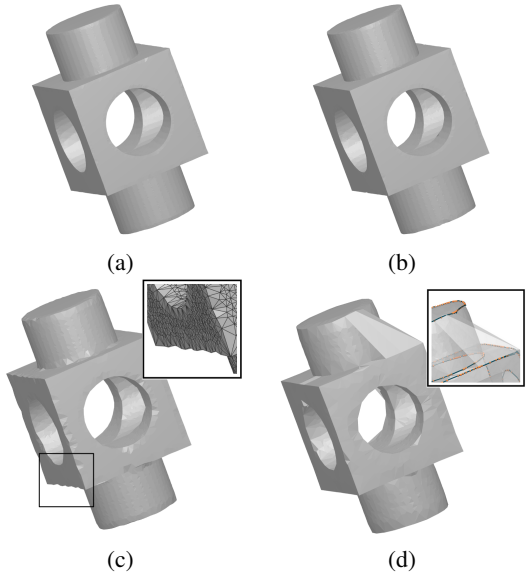


Figure 7: *The block is reconstructed from a point cloud of size (a) 74K, (b) 47K, (c) 21K and (d) 8.5K, respectively. In (d) we note that feature curves (blue curves) are still reasonably reconstructed, although not well sampled (orange dots).*

7. Limitations and Conclusion

Our 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 (convex corners usually pose no difficulty). We plan to investigate further to identify the position of these concave corners using only local information.

We will also explore how our algorithm perform on real data, such as range scans data, which can contain missing data and more general types of noise. Missing data can be challenging as the boundary of the sampling holes may not be well sampled. It will be interesting to investigate how to make our algorithm more robust for real data.

Finally, we do not yet have a theoretical guarantee for our reconstruction method. We believe that it might be possible to provide a topological guarantee for the output along the line of [CDR10], who applied weighted Delaunay refinement for meshing with theoretical guarantees.

Acknowledgement

The authors thank the anonymous reviewers for their helpful comments. Most of the models used in this paper are courtesy of AIM@SHAPE Shape Repository. The authors acknowledge the support of NSF under grants CCF-1048983, CCF-1116258 and CCF-0915996.

References

[AA06] ADAMSON A., ALEXA M.: Anisotropic point set surfaces. In *AfriGraph'06* (2006), pp. 7–13. 2

[AB99] AMENTA N., BERN M.: Surface reconstruction by Voronoi filtering. *Discrete & Computational Geometry 22*, 4 (1999), 481–504. 2

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point set surfaces. In *IEEE Visualization '01* (2001), pp. 21–28. 2

[ACDL02] AMENTA N., CHOI S., DEY T. K., LEEKHA N.: A simple algorithm for homeomorphic surface reconstruction. *Intern. J. Comput. Geom. Appl. 12*, 1–2 (2002), 125–141. 2

[ACSTD07] ALLIEZ P., COHEN-STEINER D., TONG Y., DESBRUN M.: Voronoi-based variational reconstruction of unoriented point sets. In *Sympos. Geometry Processing* (2007), pp. 39–48. 2

[ASGCO10] AVRON H., SHARF A., GREIF C., COHEN-OR D.: L_1-Sparse reconstruction of sharp point set surfaces. *ACM Trans. Graph. 29*, 5 (2010), 135:1–135:12. 2

[BC02] BOISSONNAT J.-D., CAZALS F.: Smooth surface reconstruction via natural neighbor interpolation of distance functions. *Comput. Geom.: Theory Appl.* (2002), 185–203. 2

[BN03] BELKIN M., NIYOGI P.: Laplacian Eigenmaps for dimensionality reduction and data representation. *Neural Comp 15*, 6 (2003), 1373–1396. 4

[BO05] BOISSONNAT J., OUDOT S.: Provably good sampling and meshing of surfaces. *Graphical Models 67*, 5 (2005), 405–451. 3

[BQWZ12] BELKIN M., QUE Q., WANG Y., ZHOU X.: Toward understanding complex data: graph Laplacians on manifolds with singularities and boundaries. In *Conference on Learning Theory (COLT), to appear* (2012). 1, 4, 5, 10

[CCS12] CAZALS F., COHEN-STEINER D.: Reconstructing 3d compact sets. *Comput. Geom.: Theory and Appl. 45* (2012), 1–13. 2

[CDR10] CHENG S.-W., DEY T. K., RAMOS E. A.: Delaunay refinement for piecewise smooth complexes. *Discrete & Computational Geometry 43*, 1 (2010), 121–166. 2, 3, 9

[CG06] CAZALS F., GIESEN J.: Delaunay triangulation based surface reconstruction. In *Effective Computational Geometry for Curves and Surfaces, J. Boissannat and M. Teillaud (eds.)* (2006), Spirnger Verlag, Math. and Visualization, pp. 231–276. 2

[Dey06] DEY T. K.: *Curve and surface reconstruction: Algorithms with mathematical analysis*. Cambridge University Press, New York, 2006. 1, 2

[DMSB99] DESBRUN M., MEYER M., SCHRÖDER P., BARR A. H.: Implicit fairing of irregular meshes using diffusion and curvature flow. *Computer Graphics 33*, Annual Conference Series (1999), 317–324. 4

[DTS01] DINH H. Q., TURK G., SLABAUGH G.: Reconstructing surfaces using anisotropic basis functions. In *Intern. Conf. Comput. Vision (ICCV)* (2001), pp. 606–613. 2

[DW11] DEY T. K., WANG Y.: Reeb graphs: Approximation and persistence. In *Proc. 27th Sympos. Comput. Geom.* (2011), pp. 226–235. 1, 5

[FCOS05] FLEISHMAN S., COHEN-OR D., SILVA C. T.: Robust moving least-squares fitting with sharp features. *ACM Trans. Graph. 24* (July 2005), 544–552. 3

[GP07] GROSS M., PFISTER H.: *Point based graphics*. Morgan Kauffman, Massachusetts, 2007. 1

[Gri06] GRIGOR'YAN A.: Heat kernels on weighted manifolds and applications. *Cont. Math. 398* (2006), 93–191. 4

[GSBW11] GE X., SAFA I., BELKIN M., WANG Y.: Data skeletonization via reeb graphs. In *Proc. 25th Annu. Conf. Neural Information Processing Systems (NIPS)* (2011). 1, 5

[IHOS07] II J. D., HA L. K., OCHOTTA T., SILVA C. T.: Robust smooth feature extraction from point clouds. In *Shape Modeling International* (2007), pp. 123–136. 3, 5

[JWS08] JENKE P., WAND M., STRAβER W.: Patch-graph reconstruction for piecewise smooth surfaces. In *Proceedings Vision, Modeling and Visualization (VMV)* (2008). 3

[KBH06] KAZHDAN M., BOLITHO M., HOPPE H.: Poisson surface reconstruction. In *Sympos. Geom. Processing* (2006), pp. 61–70. 2

[KWT88] KASS M., WITKIN A., TERZOPOULOS D.: Snakes: Active contour models. *International Journal of Computer Vision 1* (1988), 321–331. 6

[LCOL07] LIPMAN Y., COHEN-OR D., LEVIN D.: Data-dependent mls for faithful surface approximation. In *Sympos. Geom. Processing* (2007), pp. 59–67. 2

[MOG09] MÉRIGOT Q., OVSJANIKOV M., GUIBAS L.: Robust voronoi-based curvature and feature estimation. In *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling* (2009), SPM '09, pp. 1–12. 3

[MW11] MARTIN S., WATSON J.-P.: Non-manifold surface reconstruction from high-dimensional point cloud data. *Comput. Geom. Theory Appl. 44*, 8 (Oct. 2011), 427–441. 3

[ÖGG09] ÖZTIRELI A. C., GUENNEBAUD G., GROSS M. H.: Feature preserving point set surfaces based on non-linear kernel regression. *Comput. Graph. Forum 28*, 2 (2009), 493–501. 2

[PKG03] PAULY M., KEISER R., GROSS M. H.: Multi-scale feature extraction on point-sampled surfaces. *Comput. Graph. Forum 22*, 3 (2003), 281–290. 1, 6

[RJT*05] REUTER P., JOYOT P., TRUNZLER J., BOUBEKEUR T., SCHLICK C.: Surface reconstruction with enriched reproducing kernel particle approximation. In *Sympos. Point-Based Graphics* (2005), pp. 79–87. 2

[SYM10] SALMAN N., YVINEC M., MÉRIGOT Q.: Feature preserving mesh generation from 3d point clouds. *Comput. Graph. Forum 29*, 5 (2010), 1623–1632. 3, 5

[WCPn10] WANG W., CARREIRA-PERPINÁN M. A.: Manifold blurring mean shift algorithms for manifold denoising. In *Computer Vision and Pattern Recognition* (2010), pp. 1759 –1766. 7

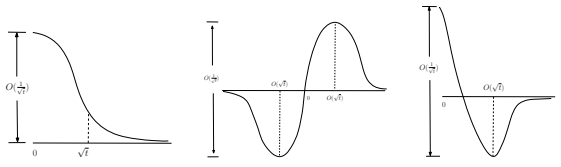
Appendix A: Graph Laplacian around Other Singularities

We now give a brief description of the behavior of the functional Laplacian on a singular surface near the three types of singularities: the boundary-feature curves, the sharp-feature curves and the intersection-feature curves. These are results of [BQWZ12], and we include a brief synopsis here for completeness and to help to provide intuition for our approaches.

Let x be a point near the singularity, and let x_0 be its nearest neighbor in the feature curves (of one of three types). In general, we have that

$$\mathcal{L}_t f(x) = \frac{1}{\sqrt{t}} D(x, x_0, t) + o(\frac{1}{\sqrt{t}}), \tag{4}$$

where $D(x, x_0, t)$ is a quantity dominated by a Gaussian (or a combination of Gaussians) whose variance is ct. In particular, setting $r = \frac{\|x-x_0\|}{\sqrt{t}}$, we have:



(a) (b) (c)

Figure 8: *The plots for the value of the term $\frac{1}{\sqrt{t}} D(x, x_0, t)$ in Eqn (4) (y-axis in the graphs) for points on and around (a) boundary, (b) intersection and (c) sharp corner singularity. In all three graphs, 0 is where the singularity locates, x-axis indicates the distance between a point to the singularity.*

(1) *boundary-feature curves:* See Figure 8 (a).

$$\mathcal{L}_t f(x) = \frac{C_1}{\sqrt{t}} e^{-r^2} + o(\frac{1}{\sqrt{t}}). \tag{5}$$

(2) *intersection-feature curves:* See Figure 8 (b).

$$\mathcal{L}_t f(x) = \frac{C_2}{\sqrt{t}} \cdot r \cdot e^{-r^2 \sin^2 \theta} + o((\frac{1}{\sqrt{t}}), \tag{6}$$

where θ is the angle between the tangent spaces of the two intersecting surface patches at x_0.

(3) *sharp-feature curves:* See Figure 8 (c).

$$\mathcal{L}_t f(x) = \frac{C_3}{\sqrt{t}} e^{-r^2} + \frac{C_4}{\sqrt{t}} \cdot r \cdot e^{-r^2 \sin^2 \theta} + o(\frac{1}{\sqrt{t}}) \tag{7}$$

Intuitively, a point around a sharp feature will both see the boundary effect and partial effect from the other surface patch.

In each case, C_1, C_2, C_3 and C_4 are proportional to a certain first order derivative of f at x_0, and is independent of t. The illustrations in Figure 8 assume that (the absolute values of) these terms are bounded from below. For certain functions, their partial derivatives may vanish and these terms C_is may be zero. However, for the specific functions (the coordinate functions) that we will use in our algorithm, these C_is are always bounded by a constant from below.

It is worth pointing out that for a point x **on** the intersection-feature curves (i.e, $x = x_0$ and $r = 0$), the $O(\frac{1}{\sqrt{t}})$ term vanishes and Eqn (6) eventually leads to that $\mathcal{L}_t f(x) = \Delta_{S_1} f(x) + \Delta_{S_2} f(x) + o(1)$, which is simply the addition of the two manifold-Laplacian $\Delta_{S_1} f(x)$ and $\Delta_{S_2} f(x)$ for the two intersecting surface patches at this point x. See also Figure 8 (b) where at the origin 0 (i.e, on the intersection singularity), the first terms becomes zero. This is different from the other two types of singularities, where the value of $\mathcal{L}_t f(x)$ is of order $\Theta(\frac{1}{\sqrt{t}})$ **at** singularities; recall Figure 8 (a) and (c). The points with high $\mathcal{L}_t f$ values are concentrated around two bands which are about $O(\sqrt{t})$ distance around the intersection singularity (i.e, the origin 0 in the picture).