Voronoi-based Feature Curves Extraction for Sampled Singular Surfaces

Tamal K. Dey, Lei Wang

The Ohio State University

Abstract

The detection and reconstruction of feature curves in surfaces from a point cloud data is a challenging problem because most of the known theories for smooth surfaces break down at these places. The features such as boundaries, sharp ridges and corners, and curves where multiple surface patches intersect creating non-manifold points are often considered important geometries for further processing. As a result, they need to be preserved in a reconstruction of the sampled surface from its point sample. The problem becomes harder in presence of noise. We propose a robust Voronoi-based pipeline that engages several sub steps consisting of approaches proposed originally for smooth case. We modify or enhance them to handle features in singular surfaces. The experimental results provide the evidence that the method is effective.

1. Introduction

The problem of extracting feature curves from point sets sampled from a two dimensional object in $\mathbb{R}^3$ that is not necessarily a smooth manifold arises in various reverse engineering applications. These objects termed as *singular surfaces* may have surface patches that meet, cross, or ends along feature curves. A point set that samples such a surface may not sample these curves exactly, but only their neighborhoods, possibly with noise. A faithful reconstruction of the surface requires detecting and reconstructing the feature curves and then incorporating them in the final reconstruction of the surface. In this paper, we propose a Voronoi-based method to detect and reconstruct the feature curves from unoriented points. These feature curves can be effectively used with a recently introduced surface reconstruction algorithm [1] to reconstruct singular surfaces while preserving their features.

1.1. Previous Works

Detect-and-connect strategy has become a standard for feature curves extraction from point clouds. The work of Gumhold et al. [2] detects feature points by individually applying principal component analysis (PCA) on a fixed size neighborhood graph for each point. Feature curves are extracted by connecting the detected feature points. Pauly et al. [3] extend this approach to a multi-scale one by automatically determining the size of the neighborhood. Later, this detect-and-connect strategy is used for feature preserving surface reconstruction. Salman et al. [4] also take the approach of PCA to find points close to features, in which they analyze the convolved Voronoi covariance matrix. Feature curves are subsequently recovered by clustering feature points and connecting a subset of them. They apply remeshing and protecting balls [5] on the reconstructed surface to incorporate the feature curves. Due to the use of implicit surfaces, their approach can not handle non-manifolds or manifolds with boundaries.

Recently, Weber et al. [6] apply Gauss map clustering [7] to find feature points and locally fit feature curves as cubic Beizer splines. Dey et al. [1] use Gaussian-weighted graph Laplacian to identify singular points and Reeb graph to connect them into feature curves. They also introduce WeightCocone, a singular surface reconstruction algorithm that can respect features.

In the context of feature preserving surface reconstruction, several strategies have been proposed without explicitly rebuilding the feature curves.

Kobbelt et al. [8] introduce a volume based reconstruction which preserves features. However, it expects the surface to be input as an implicit function, or a polygonal mesh, or a point cloud with oriented normal vectors. Wang et al. [9] propose a voxel-based surface reconstruction algorithm which can handle non-manifolds and boundaries but not sharp features.

Fleishman et al. [10] introduce robust moving least squares (MLS) which adapts the MLS technique of Alexa et al. [11] to non-smooth surfaces. This method considers the surface as a collection of piece-

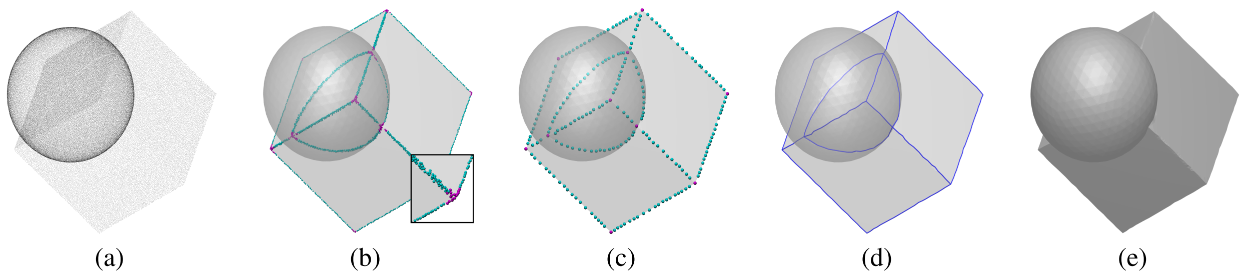


Figure 1: Work-flow of FeatureRecon: (a) Take a point cloud as input, (b) FeatureRecon detects feature points and determines their type: magenta ones are those located near corners where feature curves intersect; other feature points are colored as cyan. (c) Then FeatureRecon selects a subset of feature points and uses them to rebuild feature curves, as shown in (d). (e) As an optional step, FeatureRecon uses WeightCocone [1] to reconstruct the surface.

wise smooth patches. Each patch is approximated as a bivariate polynomial based on a subset of the sample whose residual with respect to the polynomial is smaller than a threshold. This method is robust to outliers, but may generate jagged features for sharp crease and cannot handle non-manifolds.

Lipman et al. [12] introduce singularity indicator field (SIF), which measures the proximity of each point to feature curves. The continuous SIF allows local approximation of features in a threshold-free manner. But, this method cannot handle non-manifold structures, and requires careful tuning for noisy data. Later, Ötireli et al. [13] combine implicit MLS with a statistical technique, specifically, local kernel regression, to keep features. Their method is robust to outliers and sparse sampling, but requires orientation, which could be unreliable for samples derived from non-manifolds. Moreover, the usage of the signed distance field greatly limits its handling of singularities.

Jenke et al. [14] group points into patches and normals for patches are estimated by PCA. Feature points along intersections of patches which have different normals are identified as feature points. Avron et al. [15] introduce a two step reconstruction algorithm. In the first step, orientation is corrected as a feature aware L_1-norm minimization. Then, points are moved along their normal directions so that their positions match corrected orientation.

Recently, Martin and Watson [16] introduce an incremental algorithm which can handle non-orientable and self-intersecting surfaces, but it does not preserve sharp features. Digne et al. [17] transform reconstruction problem into optimally transporting input point set to a subset of Delaunay triangulation such that features can be preserved. Their vertex relocation mechanism aims at preserving features.

Also, there are techniques that recover features from reconstructed surfaces. For example, Attene et al. [18] identify smooth areas in a triangular mesh and subdivide edges and triangles that connect different smooth areas. Features are recovered by moving those new vertices to the intersection of neighboring smooth areas.

1.2. Our method

We extend a provable Voronoi based method for dimension detection to detect feature points. We filter these points and then connect the resulting set with a curve reconstruction algorithm. These curves in conjunction with WeightCocone, a variation of the well known Cocone algorithm [19], provide a feature preserving surface reconstruction. Figure 1 shows the outline of our algorithm.

Our contributions are two folds: First, we introduce a novel feature recovery algorithm. It consists of a feature point detection step that is able to isolate points near the feature curves from a possibly noisy sample of a singular surface with no requirement of orientation. This is followed with a filtering step that connects the remaining points into piecewise linear curves approximating the original features. Second, we provide a single framework to handle a wide variety of input that include surfaces with sharp features, boundaries, and non-manifold points. The effectiveness of our feature recovery algorithm is demonstrated by the success of a feature preserving surface reconstruction algorithm that incorporates the computed feature curves.

2. Overall Algorithm

Given a point cloud P sampled from a collection of surface patches, we aim at reconstructing the feature curves, i.e., boundaries, sharp ridges, and patch intersections. To achieve it, we first detect points close to the feature curves. Then, we reconstruct feature curves from those feature points. Algorithm 1 gives the outline.

Algorithm 1 FeatureRecon($P, \rho_1, \rho_3, r_c, r_f, t$)

FeatureRecon first finds a subset of points P^* close to features from input point cloud P. Then it estimates curve direction by PCA and select a subset $P' \subseteq P^*$ to rebuild feature curves F using NNCrust.
Main parameters: r_c, ρ_1 and ρ_3 are used for feature points detection; r_f is the radius used for feature extraction; t is the distance parameter used by Filter.

```
1: $P^* :=$ FeatureTest($P, r_c, \rho_1, \rho_3$)
2: FeaturePCA($P^*, r_f$)
3: $P' :=$ Filter($P^*, r_f, t$)
4: $F :=$ NNCrust($P'$)
5: return $F$
```

FeatureTest finds feature points $P^* \subseteq P$ which are close to feature curves. FeaturePCA determines feature direction information from P^* and distinguishes corner points from other feature points. These two steps are detailed in Section 3. Next we reconstruct the feature curves. However, the point set P^* might be noisy for curve reconstruction, so we apply a filtering step to smooth out P^* and obtain a subset $P' \subseteq P^*$. We use a modified version of the well known algorithm NNCrust [20] for curve reconstruction in 3D to reconstruct the feature curves F. The modifications are warranted for handling singularities as described in Section 4.

For surface reconstruction one may call the WeightCocone algorithm described in [1] that is designed to reconstruct surfaces in presence of feature curves. For this we need to refine the feature curves in F and remove points from P that are inside the union of protecting balls centered around feature points. WeightCocone reconstructs the surface from refined feature points and the rest of the points using a weighted Delaunay triangulation.

3. Feature Points Detection

As observed in [21], points $P^* \subset P$ near the feature curves could be detected by studying the shape of their Voronoi cells. These cells differ in shape according to the dimension of the features.

For a point $p \in P$, let V_p denote its Voronoi cell in the Voronoi diagram of P. The shape of V_p reveals information about p's proximity to a feature: if p is close to sharp edges or boundaries, V_p has a thin plate shape; if p is around patch intersection, V_p is like a rounded polyhedron; otherwise, if p is on a smooth surface patch away from these features, V_p is long and slim like a pencil; see Figure 2(a) for an illustration. However, Voronoi cells can easily be distorted by noise or the global nature of the Voronoi diagram. In this paper, we propose to use the power diagram, a weighed version of the Voronoi diagram, to solve these two problems. See Figure 2 for an illustration.

3.1. FeatureTest

For a sample point $p \in P$, we examine certain subsets of the Voronoi cell V_p which we call Voronoi subpolytopes after [21]. These subpolytopes $V_p^i \subseteq V_p, i = 1, 2, 3$ are defined as follows in [21] which we reproduce here.

Assume that V_p^i is already defined. The farthest point v_p^{i+} in V_p^i from p is called the positive *pole* of V_p^i and the vector $\mathbf{v}_p^{i+} = v_p^{i+} - p$ its positive *pole vector*. In case V_p^i is unbounded, $\mathbf{v}_p^{i+}$ is taken at infinity, and the direction of $\mathbf{v}_p^{i+}$ is taken as the average of all directions given by unbounded edges.

Let $V_p^3 = V_p$. The Voronoi subpolytope V_p^{i-1} is the minimal polytope containing all points

$$\{x : \angle((x - p), \mathbf{v}_p^{i+}) = \frac{\pi}{2}\},$$

where $\angle(\mathbf{v}, \mathbf{w})$ denotes the acute angle between the lines supporting two vectors $\mathbf{v}$ and $\mathbf{w}$.

Obviously, $V_p^{i-1} \subset V_p$ is a polytope orthogonal to $\mathbf{v}_p^{i+} = v_p^{i+} - p$, and $\mathbf{v}_p^{i+}, 1 \le i \le 3$, form an orthogonal basis of V_p. The length $H_p^i = \|\mathbf{v}_p^{i+}\|$ is called the *height* of the Voronoi subpolytope V_p^i. H_p^i measures elongation of V_p^i and fatness of V_p^{i+1}.

Feature Point Condition. When there is no noise, a first approach designed after the observation about the Voronoi subpolytopes can be as follows. For a point $p \in P$, we use two parameters ρ_1 and ρ_3 to quantify the shape of V_p. If $H_p^1/H_p^3 > \rho_3$, V_p is considered short and fat and it lies close to patch intersections; otherwise if $H_p^1/H_p^2 < \rho_1$, V_p is like a thin plate, p is considered to be near a boundary or a sharp feature curve. For both cases, p can be gathered into P^*. Otherwise, p is either lying on a surface patch away from the feature curves or close to a corner where multiple feature curves meet. To handle the later case, define the negative pole v_p^{i-} of V_p^i as the farthest point in V_p^i such that the negative pole vector $\mathbf{v}_p^{i-} = v_p^{i-} - p$ makes an angle more than $\frac{\pi}{2}$ with $\mathbf{v}_p^{i+}$. If p is close to a corner, $\mathbf{v}_p^{3-}$ will be much shorter than $\mathbf{v}_p^{3+}$ and thus can be used to distinguish these two cases. Figure 3 shows Voronoi subpolytopes of boundary, corner and smooth surface point, respectively.

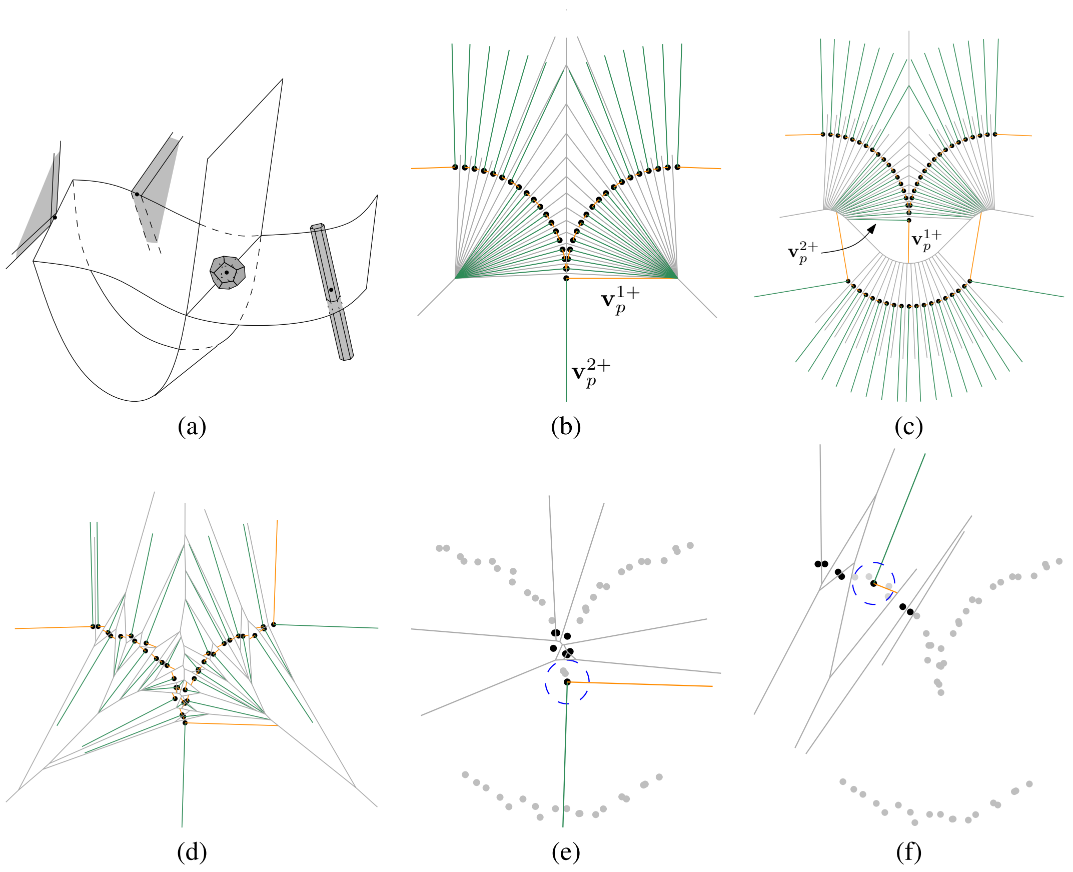
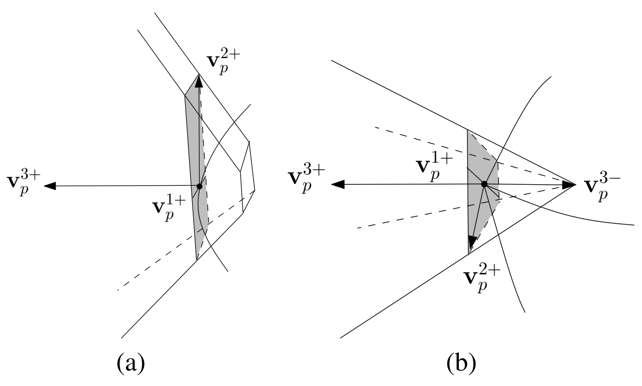


Figure 2: (a) Voronoi cells of points on various locations can have different shapes. (b) Voronoi diagram and subpolytope pole vectors for a small points set. Green and orange segments show $\mathbf{v}_p^{2+}$ and $\mathbf{v}_p^{1+}$, respectively. (c) Voronoi diagram is vulnerable to remote part: Notice how Voronoi cell of the corner point is reduced and its $\mathbf{v}_p^{2+}$ and $\mathbf{v}_p^{1+}$ are exchanged because of the invasion. (d) Voronoi diagram also can be easily distorted by noise. (e) and (f) Locally built Voronoi diagram can greatly alleviate the influence of noise and remote parts. Only black dots are inserted into the Voronoi diagram. Dashed circle demonstrates the weight.

Effects of globality and noise. The above approach faces two difficulties, one due to the global nature of the Voronoi diagram, and the other due to the presence of noise. The global geometry can influence the shape of the Voronoi cells to the extent that the above feature detection can return false identifications. As Figure 2(c) illustrates, the Voronoi cells can be invaded by remote part distorting their shapes and hence the pole vectors. We solve this problem by locally building the Voronoi diagram in the neighborhood of the point in question. Let $N(p, P, r_c)$ denote the points in P that lie within a radius r_c from a point $p \in P$. We only use points in $N(p, P, r_c)$ to compute V_p^i.

The noise can also distort the shape of the Voronoi cells to invalidate the previous feature detection conditions. As Figure 2(d) shows, even a small perturbation can change the shapes of the Voronoi cells considerably. Here we take a cue from the approach of Alliez et al. [22] who observed that, although individual cells can be unreliable, the union of Voronoi cells remains robust against the influence of noise. Based on this observation, we estimate a specific Voronoi cell by enlarging it, or equivalently increasing the influence of the point in question. The weighted Voronoi diagram provides a natural tool to achieve this goal.

Equip every point $p_i \in P$ with a weight $w_i \in \mathbb{R}$. The *power distance* between the two weighted points (p_i, w_i) and (p_j, w_j) is defined as $\|p_i - p_j\|^2 - w_i - w_j$, where $\|p_i - p_j\|$ is the Euclidian distance between p_i and p_j. The power diagram associated with the weighted point set P is a partition of $\mathbb{R}^3$ such that the power diagram cell for (p_i, w_i) consists of points with no greater power distance to any other points in P than p_i. Clearly, the power diagram is a generalization of the Voronoi diagram by changing its distance metric to the power distance. The definitions of Voronoi subpolytopes and their



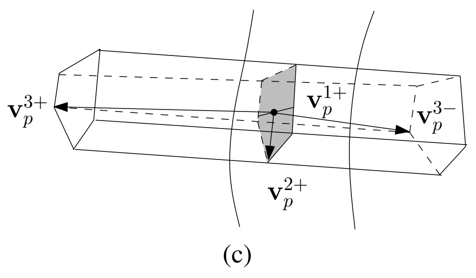


Figure 3: Voronoi subpolytopes and pole vectors of (a) boundary, (b) corner and (c) smooth surface point. Voronoi cell of p is V_p^3. Shaded polygon is V_p^2 and the segment in V_p^2 is V_p^1.

heights can be directly adapted to the power diagram. In the rest of this paper, we do not distinguish between the Voronoi diagram and the power diagram.

If we give a positive weight to p and set the weight of other points to 0, the Voronoi cell of p will enlarge because the power distance from any point in the space to p decreases while to other points remains unchanged. Imagine a weighted point (p, w) as a ball centered at p with radius $\sqrt{w}$. The points in the ball must have a non-positive distance to p while its distance to other points still remains non-negative. So the ball must be in the Voronoi cell of p. If the weight of p is larger than the scale of noise, this enlargement can expel nearby noisy points and locally eliminate the effect of noise while the shape of V_p still preserves feature information.

Leveraging the local testing and the power diagram together, our feature detection algorithm works as described in Algorithm 2.

Algorithm 2 FeatureTest(P, r_c, ρ_1, ρ_3)

```
1: $P^* := \emptyset$
2: for all $p \in P$ do
3:     $P_p := N(p, P, r_c) \setminus N(p, P, \sqrt{w})$
4:     Build Voronoi diagram of $P_p$ with $p$ weighted $w$ and other points weighted 0
5:     Compute $V_p^i$
6:     if shapes of $V_p^i$ meet the Feature Point Condition then
7:         $P^* := P^* \cup \{p\}$
8:     end if
9: end for
10: return $P^*$
```

In implementation, we set w as $r_c/4$ and points with non-positive power distance, i.e., the points inside the ball, are not inserted into the Voronoi diagram which we limit to have at most 80 points to speed up the computation.

3.2. FeaturePCA

To build feature curves from detected feature points P^*, we need to estimate the direction of the feature curves. In particular, these directions help to identify the points near corners where multiple feature curves meet. Ideally, if p is on a sharp crease, $\mathbf{v}_p^{1+}$ should indicate the local direction of the curve. However, $\mathbf{v}_p^{1+}$ can be easily distorted by slight anomalies in the neighboring points. So, we turn to a PCA based dimension test to obtain the direction information and recognize corner points, i.e., where feature curves meet.

We compute the covariance matrix M_p of $N(p, P^*, r_f)$ with an input parameter r_f. It is well known that the eigenvectors of M_p capture the spanning axes of $N(p, P^*, r_f)$, and eigenvalues of M_p capture the anisotropy of $N(p, P^*, r_f)$. Let $\lambda_1(p) \le \lambda_2(p) \le \lambda_3(p)$ be the eigenvalues of M_p and $\mathbf{v}_p$ be the eigenvector corresponding to $\lambda_3(p)$. If p is just a feature point, $N(p, P^*, r_f)$ distributes in the direction of the curve. Then, $\lambda_3(p)$ is significantly larger than $\lambda_1(p)$ and $\lambda_2(p)$ and $\mathbf{v}_p$ indicates the direction of the feature curve near p. In our implementation, if $\lambda_3(p)/\lambda_2(p) < 5$, p is considered a non-corner feature point, otherwise, p is considered a corner point. Figure 4 shows detected feature points of Carved Object and Sharp Sphere.

4. Feature Curves Reconstruction

Once P^* is computed, we need to reconstruct the feature curves from P^*. Usually P^* is a noisy sample of the curves to be reconstructed. So, we filter it before connecting remaining points with the approach of a curve reconstruction algorithm called NNCrust [20].

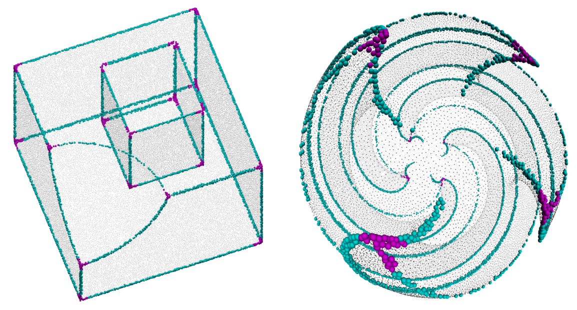


Figure 4: Detected feature points of Carved Object and Sharp Sphere overlaid with their input point clouds. Corner points are colored as magenta. Other feature points are marked as cyan.

4.1. Filter

Algorithm 3 Filter(P^*, r_f, t)

```
1: $P' := \emptyset$
2: Find groups of corner points using union-find
3: **for all** groups of corner point **do**
4:     Find the point $p$ in the group which is closest to the barycenter of the group
5:     $P' := P' \cup \{p\}$
6: **end for**
7: **for all** $p \in P^*$ where $p$ is a feature point **do**
8:     $d_p := t \times \|p - \text{barycenter}(N(p, P^*, r_f))\|$
9: **end for**
10: Sort $P^*$ by $d_p$ in ascending order
11: **for all** $p \in P^*$ and $p$ is a non-corner point **do**
12:     **if** $\min_{q \in P'} \|p - q\| > d_p$ **then**
13:         $P' := P' \cup \{p\}$
14:     **end if**
15: **end for**
16: **for all** $p \in P^* \setminus P'$ and $p$ is a non-corner point **do**
17:     **if** $\nexists q \in P' s.t. \|p - q\| < r_f/2$ **then**
18:         $P' := P' \cup \{p\}$
19:     **end if**
20: **end for**
21: **return** $P'$
```

Usually, points in P^* are distributed around feature curves, while NNCrust just needs a few of them for reconstruction. The set P' of filtered points is grown iteratively. First, we filter the corner points, and then the rest of the feature points. Figure 5 and Algorithm 3 show how filtering is performed.

For an intersection among feature curves, multiple points around it might be marked as corner points, see Figure 4. The NNCrust algorithm needs only a single point per corner, so we group neighboring corner points

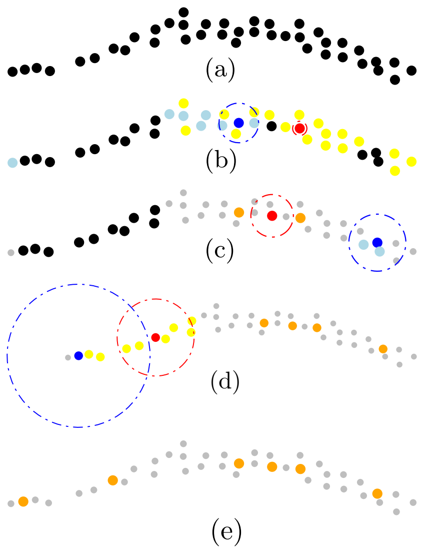


Figure 5: Illustration of how points are filtered by Filter: (a) input feature points, (b), (c), and (d) show the successive steps of filtering; in each step two points, one red and one blue, are selected. Because of the red point, the yellow points cannot be selected by the condition in step 12. Similarly, the light blue points cannot be selected because of the blue point. The orange points in each of (c) and (d) are the points selected in previous step. Dotted circles depict d_p of the selected points. (e) Output of Filter.

following a union-find strategy. Each point is initialized as a group, and the distance between two groups is measured as the distance between the closest pair of points from each group. We iteratively merge groups if the distance between two groups is less than an input parameter r_f. For each merged group only the point which is closest to the barycenter of the group is inserted into P' (steps 3-6, Algorithm 3).

For filtering the feature points other than corners, we adopt the following strategy. We keep the points that are relatively closer to the feature curves, but at the same time are mutually far enough so that a small deviation in their position is not accentuated as large relative noise during curve reconstruction. We balance these two contradicting goals as follows. We compute the distance d_p of every point p to the barycenter of a group of points around it (steps 7-9, Algorithm 3). Specifically, we compute

$$d_p = t \times \|p - \text{barycenter}(N(p, P^*, r_f))\|.$$

The barycenter should be relatively close to the feature curves. We want to keep those points whose distance to the respective barycenters are small, but no two of them should be mutually close (steps 11-15). This is achieved by sorting the points in ascending order of d_p, and then selecting only those points that are at least d_p

distance away from the rest of the selected points so far. By processing the points in ascending order of d_p, we favor the points that are closer to the feature curves, and by putting a lower bound on their distances to the rest of the selected points, we prevent them to be mutually close. To prevent sparse distribution of points on non-salient features, we salvage some of the points that are not selected in the previous step by visiting P^* in ascending order of d_p for another round. If the distance between an unsaved point p and its closest point $q \in P'$ is less than $0.5r_f$, p is salvaged (steps 16-20, Algorithm 3).

4.2. NNCrust

Now we can build feature curves from the well distributed points in P'. Following the NNCrust algorithm, each feature point $p \in P'$ is connected to its closest point $q \in N(p, P', 5r_f)$ if

$$|\cos(p - q, \mathbf{v}_p)| > \cos(\alpha) \text{ and } |\cos(p - q, \mathbf{v}_q)| > \cos(\alpha).$$

The angle condition with respect to the parameter α provides a control over the smoothness of the reconstructed curves. α is set to 0.8 by default. Note that q can be a corner point. In that case, $\mathbf{v}_q$ is not an accurate estimation of the curve direction and we take $|\cos(p-q, \mathbf{v}_p)| = 1.0$ to ensure that corner points are properly connected. The above operation may connect p to a point on only one side. To connect it to a point on the other side, we find another point $q' \in N(p, P', 5r_f)$ satisfying the same conditions as q while requiring $\cos(q - p, q' - p) < 0$.

Corner Recovery. Recall that the corner points are chosen as those closest to the barycenter of their group. However, the actual corner may not be the average. Such inaccurate selection can bring visible artifacts at curve junctions, as shown in Figure 6(a). To find a better choice from the group, we find all non-corner points connected with the current corner point and their connections with other non-corner points. Then, we choose the corner point as the one in the corner group which minimizes the sum of its distance to lines supporting those connections, as shown in Figure 6(b).

Now we have the set of feature curves F. Figure 6(c) shows extracted feature curves of Carved Object and Sharp Sphere.

5. Application to Surface Reconstruction

Our algorithm for computing feature curves can be used in conjunction with a surface reconstruction algorithm to recover surfaces from point samples while

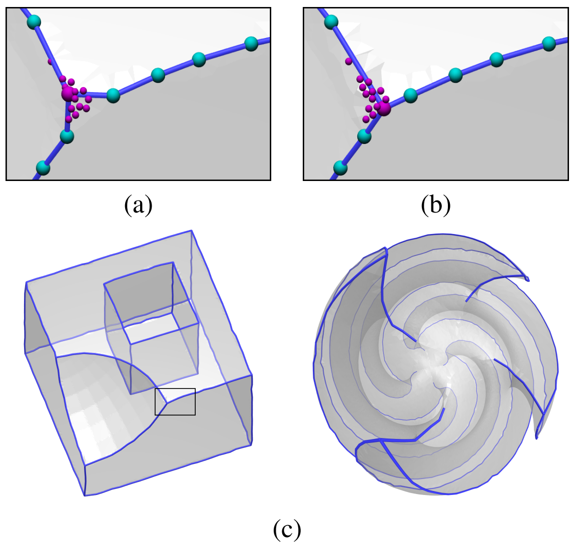


Figure 6: (a) Filter chooses corner points closest to the barycenter of their group and brings a visible artifact for Carved Object. (b) After corner recovery, the artifact is barely noticeable. (c) Recognized feature curves of Carved Object and Sharp Sphere overlaid with their reconstructed surfaces.

preserving singular features. We use the surface reconstruction algorithm recently proposed in [1]. This algorithm, named WeightCocone, modifies the original Cocone algorithm [19] to incorporate the feature curves into reconstruction. For this, it protects the feature points with protecting balls and then reconstructs using the weighted Delaunay triangulation which is the dual of the power diagram.

The output of NNCrust is actually a set of segments connected to form the feature curves. We resample these segments to get evenly placed weighted points except corners where larger protecting balls are needed to properly cover that area. Those weighted points and unweighted input points are fed to WeightCocone to get the final surface.

6. Experimental Results

FeatureRecon has been implemented as a single threaded program. We use CGAL library [23], version 3.7, to facilitate the computation of power diagrams. Feature detection and feature reconstruction stages require neighborhood computation; we use ANN library [24] to implement those queries. All experiments were performed on a laptop with 6GB RAM and a 2.27GHz processor.

We tested FeatureRecon on various clean samples with different sizes. Figure 7 shows the extracted feature point and feature curves of Wavy Saturn, Fandisk, Block and Beetle. It also highlights reconstructed surfaces with features preserved.

We compared FeatureRecon with the work of Dey et al. [1]. For Sphere Cube model in Figure 8, the method in [1] could not correctly preserve topology of feature curves. The curves generated by this method may not properly intersect, thus requiring WeightCocone to use larger protecting balls at the corner points resulting into small artifacts. FeatureRecon successfully located corners and preserved them. Figure 8 shows the reconstructed feature curves and surfaces generated by FeatureRecon and [1].

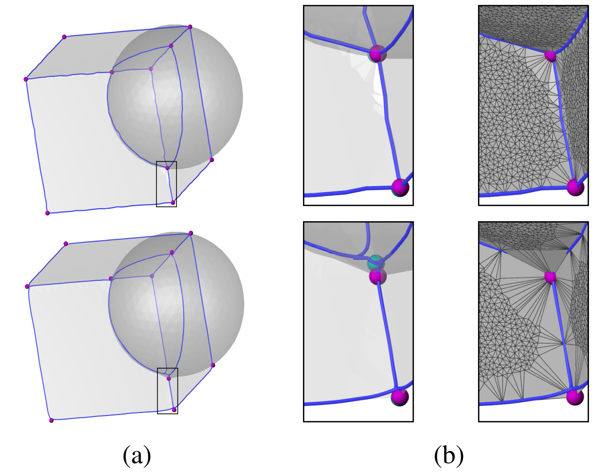


Figure 8: For Sphere Cube, FeatureRecon successfully located corners and preserved them. (a) Top: feature curves extracted by FeatureRecon. Bottom: feature curves generated by [1]; (b) Zoom in to the place where FeatureRecon correctly preserved feature curve structure. Magenta balls are recognized corner points. Actual corner points are marked as cyan to demonstrate the error. If a recognized corner points is away from its actual location, large protecting balls are needed to preserve the feature, as shown in the right column.

For sparsely sampled models like Octaflower with 39K vertices, FeatureRecon recognized most feature lines while [1] missed those near the poles. Figure 9 compares results of both methods.

In order to test the robustness of FeatureRecon to noise, we perturbed point sets of Sphere Cube and Carved Object with noise. Specifically, each point is moved towards a random direction for a maximum distance of 1% of the diagonal of the dataset bounding box. We used Mean Shift [25] to smooth the noisy point cloud and called WeightCocone to reconstruct the surface. Positive poles are provided to Mean Shift as estimated normals. As shown in Figure 10 and 11, FeatureRecon successfully preserved features for both models.

We also evaluated how FeatureRecon behaves on different level of noise. We perturbed a 42K points Smooth

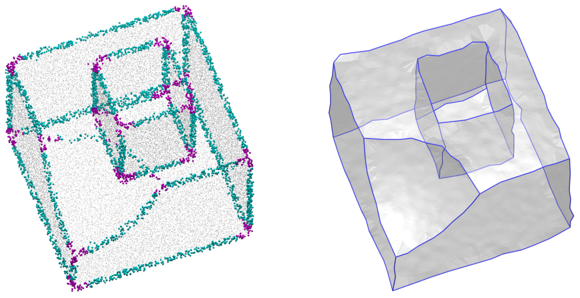


Figure 10: Left: detected feature points of Carved Object overlaid with noisy input point cloud. Right: extracted feature curves overlaid with reconstructed surface.

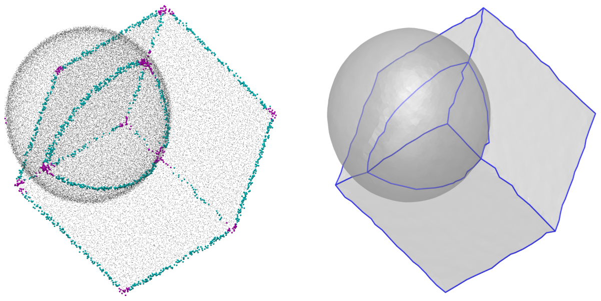
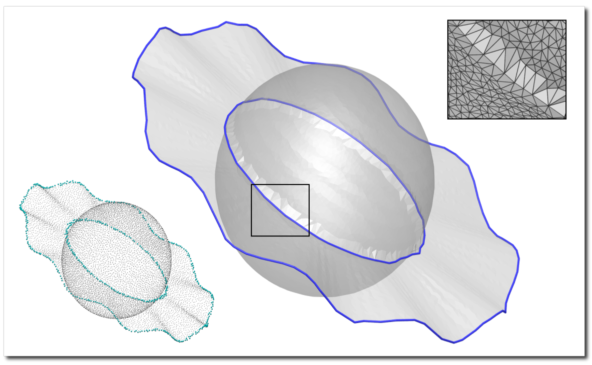


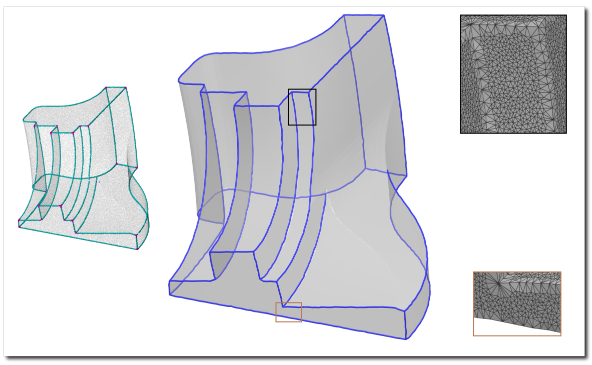
Figure 11: Extracted feature points and feature curves of noisy Sphere Cube.

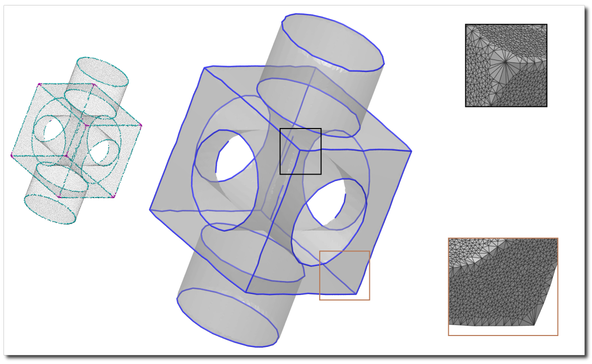
Feature model in the same way and run FeatureRecon. Bounding box of this model is $2 \times 2 \times 2.5$. We fixed $\rho_1 = \rho_3 = 0.75$, $t = 8$ and $r_f = 0.28$ for all cases and only changed r_c. Figure 12 shows the results. From the 0.5% noise case, we can see increasing r_c improves the robustness of FeatureRecon at the cost of running time. Feature point detection took 91.90, 68.71, 127.27 and 179.29 seconds for these four cases.

FeatureRecon was also tested on a range scan dataset. We downsampled Church of Lans le Villard model to 180K points and tested it. Note that this model contains noise, outliers, missing parts and its point cloud is not evenly distributed. Figure 13 shows the result.

Parameter choices and timing are reported in Table 1. The user can interactively set ρ_1 and ρ_3 in a what-you-see-is-what-you-get way to get good feature points. For clean input, r_c could be set as 4 to 8 times of the average distance between a point and its nearest neighbor. If the input is noisy, r_c should be greater than the scale of noise. The parameter r_f can be estimated based on the sampling density and the scale of the feature curves. For most cases, r_f is set between 1.4% and 4.4% of the diagonal of the bounding box to get a good neighborhood, except Beetle, Fandisk and Church, which require


(a) Wavy Saturn


(b) Fandisk


(c) Block

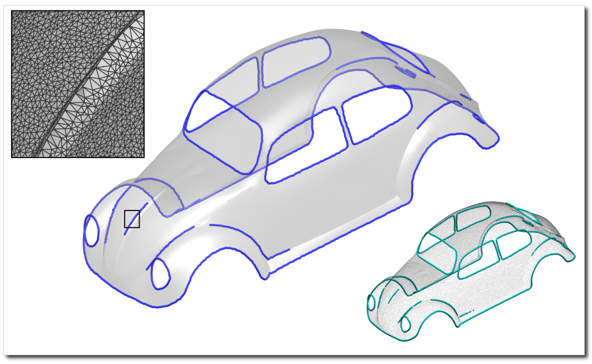
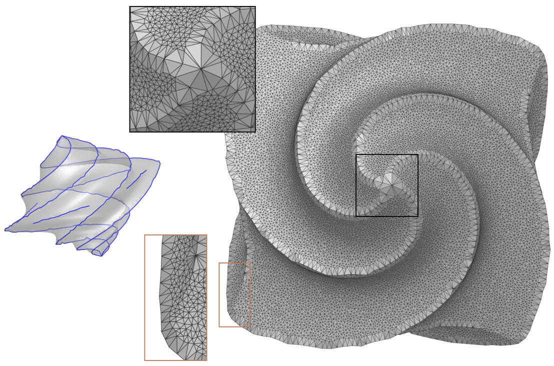

(d) Beetle

Figure 7: Detected feature points, reconstructed feature curves and shapes of various models.


(a)

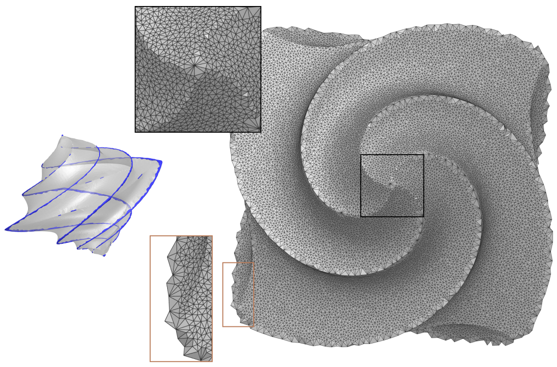

(b)

Figure 9: A sparse Octaflower model with 39K points. The method introduced in [1] failed to recognize feature curves near the poles, while FeatureRecon extracted more feature curves. (a) Feature curves and mesh generated by FeatureRecon. (b) Feature curves and mesh generated by [1].

a smaller r_f to preserve small parallel features. The parameter t depends on r_f to ensure that the filtered feature points have a good distribution. The default value $\alpha = 0.8$ works for most models except Beetle, in which case we set it to 0.6.

7. Limitations and Conclusions

We propose a feature extraction algorithm from a possibly noisy point cloud data sampled from a shape with singular features. They include surfaces with boundaries, sharp ridges, and corners, and also a collection of such intersecting surface patches. The reconstruction

Model	**Parameters**					**Timing**(sec)	
	ρ_1	ρ_3	r_c	r_f	t	(a)	(b)
Wavy Saturn(13K)	0.30	0.40	0.16/2.7%	0.2/3.3%	8	12.56	0.05
Sphere Cube(65K)	0.40	0.32	0.02/1.6%	0.036/2.9%	25	136.63	0.16
Noisy Sphere Cube(65K)	0.69	0.68	0.023/1.8%	0.053/4.2%	6	111.66	0.22
Carved Object(78K)	0.50	0.55	0.03/1.9%	0.07/4.5%	15	105.90	0.45
Noisy Carved Object(78K)	0.78	0.83	0.07/4.4%	0.12/7.5%	15	236.63	0.81
Sharp Sphere(65K)	0.70	1.00	0.5/1.4%	0.9/2.5%	35	45.55	0.48
Block(50K)	0.35	0.60	0.9/1.9%	2.0/4.2%	20	40.95	0.21
Fandisk(202K)	0.70	0.60	0.02/0.69%	0.04/1.4%	25	263.34	1.50
Octaflower(39K)	0.80	1.00	0.9/1.6%	1.3/2.3%	16	41.28	0.45
Beetle(198K)	0.50	0.45	0.8/0.47%	0.9/0.53%	20	352.2	2.00
Church(180K)	0.35	1.00	0.4/0.73%	0.5/0.91%	8	981	2.19

Table 1: Statistics of models tested by FeatureRecon. r_c and r_f are listed as the percentage of the diagonal of the bounding box for comparison. (a) and (b) are feature points detection and feature curves reconstruction, respectively.

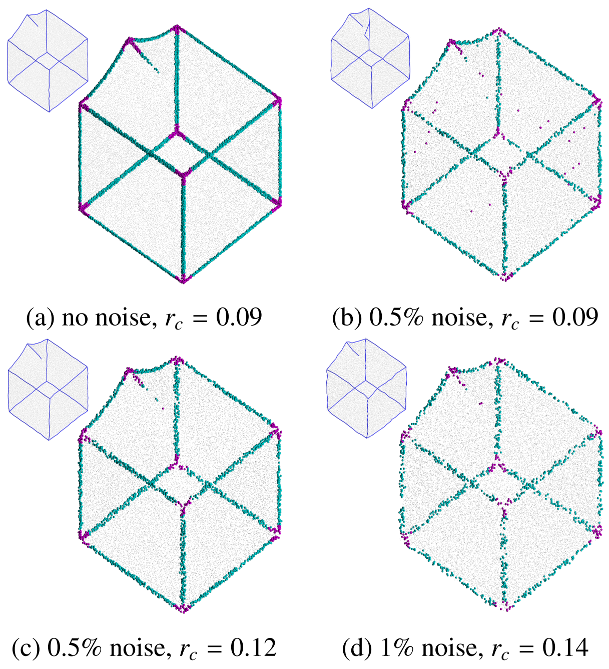

(a) no noise, r_c = 0.09
(b) 0.5% noise, r_c = 0.09
(c) 0.5% noise, r_c = 0.12
(d) 1% noise, r_c = 0.14

Figure 12: Extracted feature points and feature curves of Smooth Feature with 42K points and $2 \times 2 \times 2.5$ bounding box. All parameters are fixed except r_c. Comparing (b) and (c), we see increasing r_c could improve robustness of FeatureRecon.

preserves the features in all cases.

Our method is made more robust against noise by availing local Voronoi diagram computations for each point and also using the weighted version of the Voronoi diagram. The increased computational cost for these modifications is justified by the gain in accuracy.

The use of multiple parameters is perhaps the most serious drawback of our method. We have described how one can set these parameters. It would be nice to improve our method further so that a less number of parameters is needed. For most models, a good set of parameters can be fixed fairly quickly, so parameter tuning should not be a burden. However, we admit that for poorly sampled model like Church which contains noise, outliers, and small features close to each other, it can be difficult to find a right set of parameters that aid capturing all the features.

Although the feature detection and curve reconstruction steps are derived from provable methods, a guarantee of the entire pipeline is missing. We think it may be possible to provide theoretical guarantees for FeatureRecon, or at least for the feature detection step.

The weights used for feature detections should be at the scale of the noise. Too small a weight may not detect the feature point. Too large a weight may compromise the locality of the feature detection step and increase the computation cost. It would be interesting to come up with a method that can estimate the weights more naturally.

Acknowledgment

The authors thank the anonymous reviewers for their helpful comments. Most of the models used in this paper were obtained from the AIM@SHAPE. The Sphere

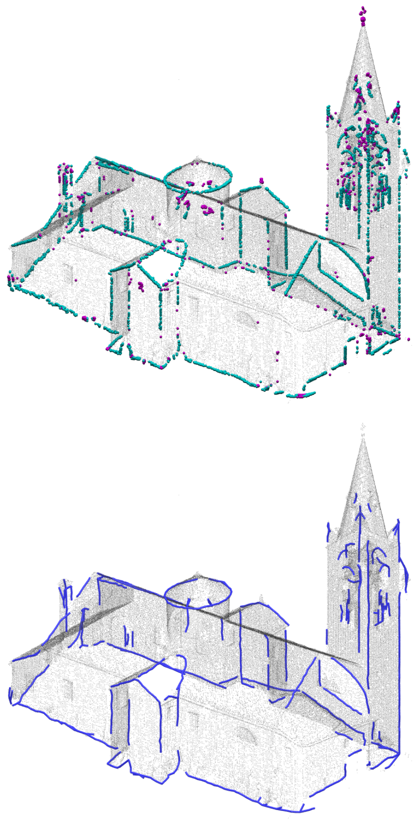


Figure 13: Feature points and feature curves of Church with 180K points.

Cube is courtesy of Xiaoyin Ge and Carved Object is courtesy of Philipp Jenke. The authors acknowledge the CGAL consortium for making the power diagram code available for experiments. This research is supported by NSF grants CCF-1064416 and CCF-1116258.

References

[1] T. K. Dey, X. Ge, Q. Que, I. Safa, L. Wang, Y. Wang, Feature-preserving reconstruction of singular surfaces, Computer Graphics Forum 31 (5) (2012) 1787–1796.

[2] S. Gumhold, X. Wang, R. MacLeod, Feature extraction from point clouds, in: Proc. Int'l Meshing Roundtable, 2001, pp. 293–305.

[3] M. Pauly, R. Keiser, M. Gross, Multi-scale feature extraction on point-sampled surfaces, Computer Graphics Forum 22 (3) (2003) 281–289.

[4] N. Salman, M. Yvinec, Q. Mérigot, Feature preserving mesh generation from 3d point clouds, in: Symp. on Geometry Processing, 2010, pp. 1623–1632.

[5] S.-W. Cheng, T. K. Dey, J. A. Levine, A practical delaunay meshing algorithm for a large class of domains, in: Meshing Roundtable, 2007, pp. 477–494.

[6] C. Weber, S. Hahmann, H. Hagen, G.-P. Bonneau, Sharp feature preserving mls surface reconstruction based on local feature line approximations, Graphical Models 74 (6) (2012) 335–345.

[7] C. Weber, S. Hahmann, H. Hagen, Sharp feature detection in point clouds, in: Shape Modeling International, 2010, pp. 175–186.

[8] L. P. Kobbelt, M. Botsch, U. Schwanecke, H.-P. Seidel, Feature sensitive surface extraction from volume data, in: SIGGRAPH, 2001, pp. 57–66.

[9] J. Wang, M. M. Oliveira, A. E. Kaufman, Reconstructing manifold and non-manifold surfaces from point clouds, in: IEEE Visualization, 2005, pp. 415–422.

[10] S. Fleishman, D. Cohen-Or, C. T. Silva, Robust moving least-squares fitting with sharp features, ACM Trans. Graph. 24 (2005) 544–552.

[11] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, C. T. Silva, Point set surfaces, in: IEEE Visualization, 2001, pp. 21–28.

[12] Y. Lipman, D. Cohen-Or, D. Levin, Data-dependent mls for faithful surface approximation, in: Symp. on Geometry Processing, 2007, pp. 59–67.

[13] C. Öztireli, G. Guennebaud, M. Gross, Feature preserving point set surfaces based on non-linear kernel regression, Computer Graphics Forum 28 (2) (2009) 493–501.

[14] P. Jenke, M. Wand, W. Straßer, Patch-graph reconstruction for piecewise smooth surfaces, in: Proceedings Vision, Modeling and Visualization, 2008, pp. 3–12.

[15] H. Avron, A. Sharf, C. Greif, D. Cohen-Or, l_1-sparse reconstruction of sharp point set surfaces, ACM Trans. Graph. 29 (5) (2010) 135:1–135:12.

[16] S. Martin, J.-P. Watson, Non-manifold surface reconstruction from high-dimensional point cloud data, Computational Geometry 44 (8) (2011) 427–441.

[17] J. Digne, D. Cohen-Steiner, P. Alliez, F. de Goes, M. Desbrun, Feature-preserving surface reconstruction and simplification from defect-laden point sets, Journal of Mathematical Imaging and Vision 45 (2013) 1–14.

[18] M. Attene, B. Falcidieno, J. Rossignac, M. Spagnuolo, Sharpen&bend: Recovering curved sharp edges in triangle meshes produced by feature-insensitive sampling, IEEE Trans. Visualization and Computer Graphics 11 (2) (2005) 181–192.

[19] N. Amenta, S. Choi, T. K. Dey, N. Leekha, A simple algorithm for homeomorphic surface reconstruction, in: Symp. on Computational Geometry, 2000, pp. 213–222.

[20] T. K. Dey, P. Kumar, A simple provable algorithm for curve reconstruction, in: Symp. on Discrete Algorithms, 1999, pp. 893–894.

[21] T. K. Dey, J. Giesen, S. Goswami, W. Zhao, Shape dimension and approximation from samples, in: Symp. on Discrete Algorithms, 2002, pp. 772–780.

[22] P. Alliez, D. Cohen-Steiner, Y. Tong, M. Desbrun, Voronoi-based variational reconstruction of unoriented point sets, in: Symp. on Geometry Processing, 2007, pp. 39–48.

[23] Cgal, Computational Geometry Algorithms Library (Version 3.7).
URL `http://www.cgal.org`

[24] Ann, A libray for Approximate Nearest Neighbor searching.
URL `http://www.cs.umd.edu/~mount/ANN`

[25] W. Wang, M. A. Carreira-Perpinán, Manifold blurring mean shift algorithms for manifold denoising, in: Computer Vision and Pattern Recognition, 2010, pp. 1759 –1766.