The Visual Computer
https://doi.org/10.1007/s00371-021-02255-7

ORIGINAL ARTICLE

# Denoising with discrete Morse theory

Soham Mukherjee[1]

Accepted: 6 July 2021

**Abstract**
Denoising noisy datasets is a crucial task in this data-driven world. In this paper, we develop a persistence-guided discrete Morse theoretic denoising framework. We use our method to denoise point-clouds and to extract surfaces from noisy volumes. In addition, we show that our method generally outperforms standard methods. Our paper is a synergy of classical noise removal techniques and topological data analysis.

**Keywords** Persistent homology · Discrete Morse theory · Topological data analysis · Noise removal

## 1 Introduction

Noise in data is a common and unavoidable problem. A typical implicit assumption when discussing noise is that the data in question is sampled from a hidden space called the *signal* or *ground truth*. Under this assumption, *noise* is deviation from the signal due to measurement error. A classic example of noisy datasets is point-clouds that are acquired via scanners. Noise is introduced in the models because of imperfections in the scanners [5]. To approximate the signal, it is imperative that the data is sufficiently dense near the signal. Otherwise, we may lose important features to noise. In this paper, we present a denoising technique based on *discrete Morse theory* [26] and *persistent homology* [21], two of the pillars of topological data analysis (TDA). Topological features obtained via persistent homology enjoy "nice" properties such as robustness to noise and scale invariance [12]. Exploiting these properties is a natural starting point in developing a denoising framework.

There is extensive literature on denoising, and a complete overview of the subject is beyond the scope of this paper. Interested readers may consult [28,41] for a detailed survey on removing "outliers" from noisy point-clouds. Our paper focuses on utilizing techniques from TDA to denoise noisy point-clouds. Notably, applications of Morse theory in computer graphics can be traced back to [35,36]. In [35], Shinagawa et al. gave an algorithm to compute *Reeb graph* from cross section of 3D images, and in [36], they used Morse theory to encode a surface. The discrete combinatorial counterpart of classical Morse theory is due to Forman [24]. We refer to [13] for an in-depth survey of techniques rooted in Morse theory, discrete Morse theory and its applications in computer graphics. It is important to point out that the use of TDA in removing noise is not new. In [7], Bremer et al. used persistence guided simplification of Morse–Smale complexes to extract topologically valid approximations; however, our goal is different since we do not aim to compute the full Morse–Smale complex. In [8], Buchet et al. described an advanced *Parfree-Declutter* algorithm and drew topology inference with the help of TDA. Our denoising and reconstruction approach is more aligned with merging approaches mentioned in [4,11,16]. In [16], Dey et al. used *heat kernel signatures* on segmented noise-free meshes for shape matching but not for denoising. In [4], the *region-growing* algorithm was used for point-cloud segmentation. In this paper, we present a modified and simplified version of their merging algorithm that does not explicitly maintain a record of merged regions. We mention two new domains where benefits of the region-merging algorithm can be leveraged.

*Our Contribution:* We provide a discrete Morse theoretic variation of the region-merging algorithm in [16]. Armed with the new understandings, e.g., the connection between persistent homology and discrete Morse theory, we present two new areas where Algorithm 1 can outperform standard approaches such as CGAL's outlier removal (Sect. 5.1), *thresholding*, and *Isosurface detection*[1] (Sect. 5.2).

✉ Soham Mukherjee
mukher26@purdue.edu

[1] Department of Computer Science, Purdue University, West Lafayette, IN 47906, USA

[1] Isosurface detection constructs an isosurface through all data points that have the same given scalar value $i$.

Published online: 18 July 2021

## 2 Background and notations

### 2.1 Discrete Morse theory

We begin by reviewing discrete Morse theory. We encourage the unfamiliar reader to consult [25,26].

**Definition 1** (Simplex, Simplicial Complex) A *k-simplex* $\sigma = \texttt{ConvexHull}\{v_0, v_1, ..., v_k\}$ is the convex hull of $k+1$ affinely independent points. These $k+1$ points are *vertices* of the k-simplex $\sigma$. The nonnegative integer $k$ is the dimension of $\sigma$. A simplex $\tau = \texttt{ConvexHull}\{v_{i_1}, \ldots v_{i_j}\}$ is a *face* of $\sigma$ if each $v_{i_\ell}$ is a vertex of $\sigma$. The simplex $\tau$ is a *facet* of $\sigma$ if the dimension of $\tau$ is $k-1$. We denote this relationship as $\tau < \sigma$. A *simplicial complex* $\mathcal{K}$ is a set of simplices such that the following conditions hold:

1. Every face of a simplex $\sigma \in \mathcal{K}$ is also in $\mathcal{K}$
2. If $\sigma_1 \in \mathcal{K}$ and $\sigma_2 \in \mathcal{K}$ intersect, then the intersection must be a face of both $\sigma_1$ and $\sigma_2$.

**Definition 2** (Morse Pairing) A discrete (gradient) vector is a pair of simplices $(\sigma, \tau)$ such that $\sigma < \tau$. A Morse pairing on $K$, denoted $M(K) = \{(\sigma, \tau)\}$, is a set of discrete vectors where each simplex in $K$ appears in at most one pair.

Given a Morse pairing $M(K)$, a simplex $\sigma \in K$ is *critical* if it does not appear in any pair in $M(K)$.

**Definition 3** (V-Path) A sequence of simplices $\tau_0, \sigma_1, \tau_1, \sigma_2, \tau_2, ..., \sigma_n, \tau_n, \sigma_{n+1}$ is a V-path if for each $i \in \{0, 1, \ldots, n\}$, $(\sigma_i, \tau_i) \in M(K)$ and $\sigma_{i+1} < \tau_i$.

This V-path is trivial if $n = 0$ and acyclic if $(\sigma_{n+1}, \tau_0) \notin M(K)$. We say that an acyclic V-path is a *gradient path*.

**Definition 4** (Discrete Gradient Vector Field) A Morse pairing $M(K)$ is a discrete gradient vector field if all V-paths induced by $M(K)$ are acyclic.

Intuitively, one can think of critical $k$-simplices in the discrete Morse setting as index-$k$ critical points in the smooth setting. For example, for a function on $\mathbb{R}^3$, critical $0-$, $1-$, $2-$ and $3-$simplices in the discrete Morse setting correspond to minima, 1-saddles, 2-saddles and maxima in the smooth case, respectively.

### 2.2 Morse cancellation and persistence pairings

A common operation in discrete Morse theory is *cancelling* critical simplices. A pair of critical simplices $(\sigma, \tau)$ with $dim(\tau) = dim(\sigma) + 1$ are *cancellable* if there is a unique V-path $\tau_0, \sigma_1, \ldots, \tau_n, \sigma_{n+1}$ where $\tau_0 = \tau$ and $\sigma_{n+1} = \sigma$. To cancel such a pair of cancellable simplices, we modify the vector field $M(K)$ by removing the discrete vectors $(\sigma_i, \tau_i)$, for $i = 1, ..., n$, and adding new discrete vectors $(\sigma_i, \tau_{i-1})$, for $i = 1, \ldots, n+1$. Effectively, this reverses the direction of the gradient path. Furthermore, following cancellation, the simplices $\sigma$ and $\tau$ are no longer critical as they each belong to a discrete vector (for example, see Fig. 1). The uniqueness of a V-path between the critical simplices $\tau$ and $\sigma$ ensures that after cancellation is performed on the pair $(\sigma, \tau)$, the gradient vector field remains acyclic.

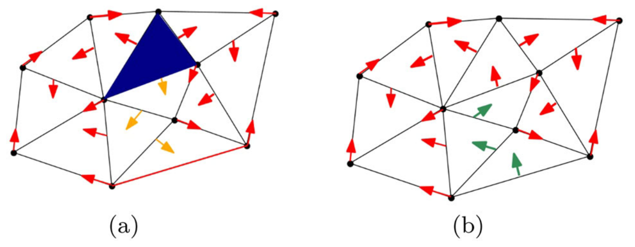

**Fig. 1** Cancellation procedure for a $2D$ toy-example. (**a**) The critical triangle $t$ (blue) can be cancelled with the critical edge $e$ (red) via the orange V-path. (**b**) The V-path is reversed (now green). Notice that $t$ and $e$ are not critical anymore

Notice that we can iteratively perform Morse cancellation on pairs of critical simplices as long as they are cancellable. One difficulty is that there exists no canonical order in which the cancellation should proceed. A popular way is to use the persistence-based approach from [2,3,15,20,38] and later used in [17–19]. We follow the steps of [3], but note that our end goal is different. Instead of creating the discrete Morse vector field, we draw its connection to region-merging algorithm. Persistent homology is a well-developed theory, and a complete exposition is well outside the scope of this paper. In this paragraph, we will give a brief outline of *persistence pairings*. Given a simplicial complex $\mathcal{K}$, we can order all the simplices $\sigma_0, \sigma_2, ..., \sigma_{N-1}$ so that all faces of a simplex $\sigma_i$ appear before $\sigma_i$ in the order. This ordering gives a (simplex-wise) filtration:

$$\mathcal{F}(\mathcal{K}) : \phi = \mathcal{K}_0 \subset \mathcal{K}_1 \subset ... \subset \mathcal{K}_{N-1} = \mathcal{K}$$

where $\mathcal{K}_i$ is the subcomplex of $\mathcal{K}$ consisting of simplices $\sigma_0, \sigma_1, ..., \sigma_{i-1}$. If one changes the value of $i$, the homology of $\mathcal{K}_i$ will change. That is, the homology of $\mathcal{K}_i$ and $\mathcal{K}_j$ are in general not the same. If one starts by considering $\mathcal{K}_0$ and consecutively considers $\mathcal{K}_1, \mathcal{K}_2$, etc., one will observe the *birth* and *death* of homology classes. Each homology class will be created by a simplex $\sigma$, and, if it dies, it is killed by a simplex $\tau$. We call $\sigma$ a *creator* or *positive* simplex, and we call $\tau$ a *destroyer* or *negative* simplex. Because $\sigma$ creates a class that is destroyed by $\tau$, we pair $\sigma$ with $\tau$. This pairing is the *persistence pairing* given by $\mathcal{F}(K)$. Computing this pairing is at the core of persistent homology (see [22] for the original algorithm).

Often, we associate some function $f : \mathcal{K} \to \mathbb{R}$ with the complex $\mathcal{K}$. We assume that $f$ has the property that if

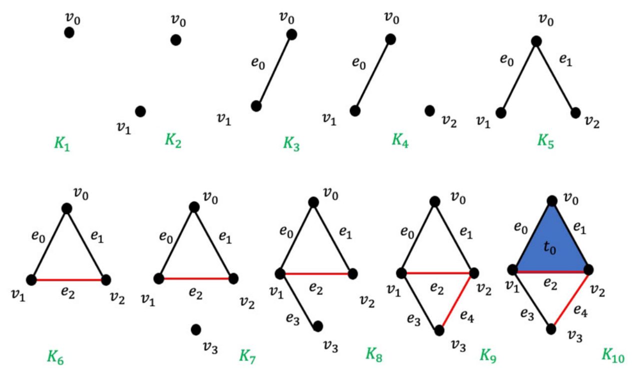

**Fig. 2** Steps of a simplex-wise filtration. Each negative edge (black) kills a connected component created by a vertex. Each positive edge (red) creates a cycle that is not a boundary. In $K_{10}$, when the triangle $t_0$ comes into the filtration it kills the homology class created by $e_2$ yielding a persistence pair $(e_2, t_0)$. Furthermore, the homology class created by $e_4$ never dies and we get a persistence pair $(e_4, \infty)$

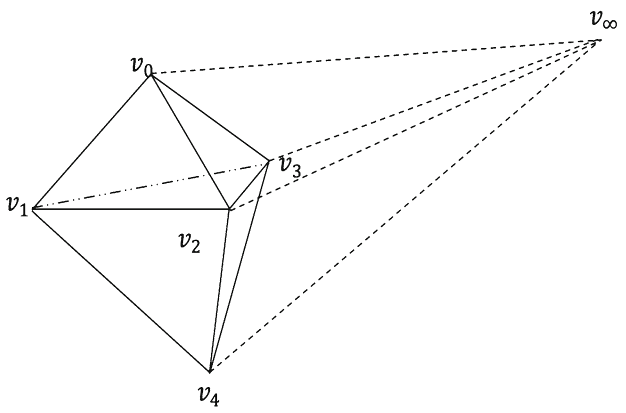

**Fig. 3** Boundary faces incident to tetrahedron meet at infinity

$\sigma$ is a face of $\tau$, then $f(\sigma) \leq f(\tau)$. Note that in the previous case, we can find such a function $f$ by taking $f$ to the index of the complex $\mathcal{K}_i$ where $\sigma \in \mathcal{K}_i \setminus K_{i-1}$. In general, we can use the function $f$ to obtain an alternative filtration $\mathcal{F}(\mathcal{K}, f) = \{\mathcal{K}_a\}_{a\in\mathbb{R}}$, where $\sigma \in \mathcal{K}_a$ if $f(\sigma) \leq \sigma$. We can compute a pairing for $\mathcal{F}(\mathcal{K}, f)$ in the same way as before. Each homology class $c$ that is born and killed in $\mathcal{F}(\mathcal{K}, f)$ then corresponds to a pair $(\sigma, \tau)$, where $\sigma$ creates the class and $\tau$ destroys it. We then say that the *persistence* of $c$ is given by $\mathsf{Pers}(c) := f(\tau) - f(\sigma)$. In the interest of brevity, we let $\mathsf{Pers}(c) = \mathsf{Pers}(\sigma) = \mathsf{Pers}(\tau)$. In the case where a class $c$ is created by a simplex $\tau$, but is never killed, then we say that $\mathsf{Pers}(c) = \mathsf{Pers}(\sigma) = \infty$.

Although we have discussed persistence homology in this section, we will not calculate all persistence pairs explicitly. As explained in [16], the region-merging algorithm implicitly computes the persistence pairs in the top dimension (in our case, triangle–tetrahedra pairings). By extension, our discrete Morse cancellation of saddle–maximal pairs will be guided by this persistence pairing.

## 3 Denoising algorithm

Let us assume we are given a domain $\mathcal{D}$ and an intensity function $I : \mathcal{D} \to \mathbb{R}$. Our algorithm works on the discrete setting, and thus, it operates on a triangulation (tetrahedralization) $\mathcal{K}$ of $\mathcal{D}$ with the piecewise-constant interpolation of $I$ as $I : \mathcal{K} \to \mathbb{R}$, defined by $I(\sigma_i) = \max\{I(u_i) : u_i \in Vert(\sigma_i)\}$ for every simplex $\sigma_i \in \mathcal{K}$. Similar to [16], our goal is to perform Morse cancellations of saddle-maxima, *i.e.,* triangle–tetrahedron pair, up to some threshold $\delta$ that as a by-product provides us a decomposition of $\mathcal{K}$ into regions. The boundary of these regions corresponding to these persistent critical tetrahedra is exactly the surface we wish to recover.

**Algorithm 1** Region Grow

```
Input: $\mathcal{K}$: Triangulated Domain, Intensity function $I : \mathcal{K} \to \mathbb{R}$, threshold $\delta$
Output: $\mathcal{E}$: Region Boundaries
1: begin
2:     $\mathcal{F} \leftarrow$ KRUSKAL-LIKE MAX-SPN TREE($\mathcal{K}$,$-I$, $\delta$)
3:     $\mathcal{E} \leftarrow$ COLLECT-REGION-BD($\mathcal{K}$,$\mathcal{F}$)
4:     return $\mathcal{E}$
5: end
```

We begin by sorting 2-simplices of $\mathcal{K}$ in decreasing order with respect to $-I$ breaking ties arbitrarily. Let $\mathcal{F} = \{t_0, t_1, \ldots, t_n\}$ be the totally ordered set of 2-simplices after sorting. We can perform the maxima–saddle cancellation with a Kruskal-like minimum spanning tree algorithm [2,3,16,17], but as we are computing persistence given by triangle–tetrahedron pairs, we instead compute a maximum spanning tree [14]. Note that effectively we are computing maximum spanning tree of the dual graph $\mathcal{G}$ of $\mathcal{K}$ where tetrahedra are nodes and triangles are edges connecting the nodes that correspond to the tetrahedra adjoining the triangles [31]. For now assume that we are dealing with manifold without boundaries *i.e.,* every triangle is incident to two tetrahedra.

Consider a generic step when an edge $e \in \mathcal{G}$ is introduced in the maximum spanning forest that joins two forests rooted at node $v_0$ and $v_1$. For the new tree, we will choose the root as node having larger functional value and the edge $e$ pairs with the node having smaller functional value. We define the persistence of the edge as $p(e) = \min\{I(v_0), I(v_1)\} - I(e)$. In the primal sense when a tetrahedron is introduced, it kills a 2-cycle and we are choosing to kill the class of youngest 2-cycle created by the triangle primal to the edge $e$. Similar to [17], if persistence of the edge is greater than $\delta$, two trees are not joined but are connected symbolically in order to compute persistence of the remaining edges. In algorithmic terms, we are maintaining two disjoint-set (Union-Find in some literature) data structures: one that helps us to compute persistence

**Algorithm 2** Kruskal-Like Maximum Spanning Tree

```
Input: $\mathcal{K}$: Triangulated Domain, Intensity function $I : \mathcal{K} \to \mathbb{R}$, threshold $\delta$
Output: Spanning Forest $\mathcal{F}$ on dual of $\mathcal{K}$
1: procedure KRUSKAL- LIKE MAX- SPN TREE
2:     $\mathcal{C} \leftarrow$ 3-Simplices (Tetrahedra) of $\mathcal{K}$
3:     $\mathcal{T} \leftarrow$ 2-Simplices(Triangles) of $\mathcal{K}$
4:     $\widetilde{\mathcal{T}} \leftarrow$ Sort $\mathcal{T}$ in non-increasing order of $I(t)$ with $t \in \mathcal{T}$
5:     for all $t \in \widetilde{\mathcal{T}}$ do
6:         $(c_0, c_1) \leftarrow$ Tetrahedra sharing $t$ as their boundary
7:         $root_0 \leftarrow find\_in\_D_1(c_0)$
8:         $root_1 \leftarrow find\_in\_D_1(c_1)$
9:         if $Root_0 \neq Root_1$ then
10:            $pers(t) \leftarrow min\{I(Root_0), I(Root_1)\} - I(t)$
11:            if $pers(t) < \delta$ then
12:                $Root\_Region_0 \leftarrow find\_in\_D_2(t_0)$
13:                $Root\_Region_1 \leftarrow find\_in\_D_2(t_1)$
14:                $merge\_in\_D_2(Root\_Region_0, Root\_Region_1)$
15:            end if
16:            $merge\_in\_D_1(Root_0, Root_1)$
17:        end if
18:    end for
19:    return $\mathcal{F}$
20: end procedure
```

of the triangle–tetrahedron pair and another to maintain the critical tetrahedra corresponding to regions. It is important to note that in Algorithm 2 the usual *Find* and *Merge* operations are on two different disjoint-set data structures $D_1$ and $D_2$. At the end of KRUSKAL-LIKE MAX-SPN TREE procedure (Algorithm 2), we are left with spanning forest $\mathcal{F}$ disconnected by edges that have persistence $\geq \delta$ and each tree is rooted at a vertex $v$. These vertices are the persistent maxima that remain critical after Algorithm 2 terminates. In the primal sense, the components of the spanning forest are the regions whose boundary we seek and conveniently enough every one of the regions is uniquely identified by its root tetrahedron (see Sect. 4). The last step of Algorithm 1 involves collection of the region boundaries. Algorithm 3 does it by iterating over all the edges again and checking whether it is incident to trees that have different roots. If an edge satisfies this condition, we add that in our output set $\mathcal{E}$ and discard it if it does not.

We conclude this section by removing the assumption that our domain is a triangulation of manifold without boundary. Essentially for a manifold with boundary, triangles situated at the boundary can be thought as incident to a tetrahedron whose functional value is very high. So for each edge dual to a boundary triangle we associate a special dummy node, $v_\infty$ *s.t.* $I(v_\infty) = \infty$. The procedure is depicted in Fig. 3.
*Time Complexity:* Since the main step of Algorithm 1 is similar to Kruskal's minimum spanning tree, time complexity of our algorithm is $\mathcal{O}(T \log T)$, where $T$ is the number of triangles present in the simplicial complex $\mathcal{K}$.

**Algorithm 3** Collect-Output($\mathcal{F}$)

```
Input: $\mathcal{T}$: Facets of Simplicial Complex $\mathcal{K}$, Spanning Forest $\mathcal{F}$
Output: $\mathcal{E}$: List of facets Separating Regions
1: procedure COLLECT- REGION- BD($\mathcal{K}$,$\mathcal{F}$)
2:     $\mathcal{E} = \phi$
3:     for all $t \in \mathcal{T}$ do
4:         $(t_0, t_1) \leftarrow$ Cells sharing $f$ as their boundary
5:         $Root_0 \leftarrow find\_in\_D_2(c_0)$
6:         $Root_1 \leftarrow find\_in\_D_2(c_1)$
7:         if $Root_0 \neq Root_1$ then
8:             $\mathcal{E} \leftarrow \mathcal{E} \cup \{t\}$
9:         end if
10:    end for
11:    return $\mathcal{E}$
12: end procedure
```

## 4 Explanation and proofs

Let $\mathcal{R}(c)$ denote the set of tetrahedron that can be reached by following the tetrahedron–triangle V-path starting from critical tetrahedron $c$. We show that Algorithm 1 computes boundary of the region, $\partial\mathcal{R}(c)$, where $c \in$ {Critical tetrahedra} and $pers(c) \geq \delta$. The following lemma summarizes results of [3,19].

**Lemma 1** *The following statement holds for $\mathcal{F}$, output of Algorithm 2 w.r.t. any finite $\delta > 0$.*

1. *$\mathcal{F}$ consists of multiple spanning trees $\{T_1, T_2, \ldots, T_k\}$.*
2. *Each spanning tree $T_i$ is rooted at node $v_i$ that corresponds to critical tetrahedron $c_i$ with $pers(c_i) \geq \delta$.*

Note that when we are merging two regions, $\mathcal{R}(c)$ and $\mathcal{R}(c')$ via a critical triangle $t \in \partial\mathcal{R}(c) \cap \partial\mathcal{R}(c')$, essentially we are performing Morse cancellation of the pair $t$ and $argmin_{x \in \{c,c'\}} I(x)$ assuming the existence of an intensity function $I$. In case of a tie, between $c$ and $c'$ we choose the tetrahedron that comes later in the filtration to cancel with $t$. Instead of maintaining the triangle–tetrahedron V-paths explicitly, we are keeping record of the tetrahedra belonging to a V-path emanating from a critical tetrahedron by connected components with the help of disjoint-set data structure.

**Lemma 2** *At any generic step if $c_i$ is a critical tetrahedron then $\partial\mathcal{R}(c_i)$ is the boundary of the tetrahedra that can be reached following the triangle tetrahedron V-path starting from $c_i$.*

We prove this with induction on the steps of Algorithm 2. At the beginning, every 2-simplex (triangle) and 3-simplex (tetrahedron) is critical. In terms of gradient vector field, there does not exist any V-path. Consider the first iteration of Algorithm 1. Let $c_0$ and $c_1$ are the coface of the triangle $t_0$ picked by the algorithm. As we have not performed any merge operation yet, both $find\_in\_D_1(c_0)$, $find\_in\_D_2(c_0)$ and

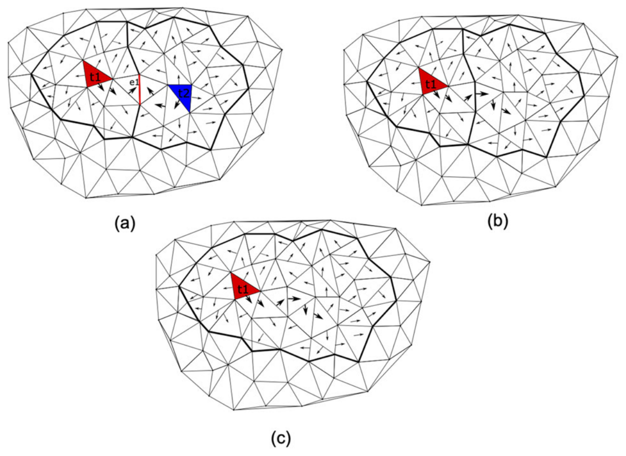

Fig. 4 2D analogue of the merging procedure. (**a**) $t_1$ (red), $t_2$ (blue) are critical triangles and $e_1$ (red) is a critical edge. (**b**) $e_1$ merges with $t_2$. (**c**) Updated region boundary

$find\_in\_D_1(c_1)$ , $find\_in\_D_2(c_1)$ yield $c_0$ and $c_1$, respectively. Assume that $pers(t_0) < \delta$, Root is chosen to be $c_0$ and $t_0$ is merged with $c_1$. As a result of this, we have created a V-path $c_0, t_0, c_1$; more specifically, we have cancelled the pair $(t_0, c_1)$. Note that after this step $t_0$ and $c_1$ are not critical anymore. Let at any point of time $\mathcal{R}(c)$ denote the region corresponding to the tetrahedron $c$ and $\partial\mathcal{R}(c)$ its boundary. At first, $\forall c \in \mathcal{C}, \partial\mathcal{R}(c_i) = \partial c_i$. After first iteration $\mathcal{R}(c_0)$ becomes $\partial\mathcal{R}(c_0) = \partial c_0 \cup \partial c_1 \setminus \{t_0\}$. Generalizing this we obtain that, at any generic step for a critical tetrahedron $c$,

$$\partial\mathcal{R}(c) = \bigoplus_v \partial c_v$$

where

$$c_v = \{c_i \in \mathcal{C} : c_i \in \text{V-Path from critical tetrahedron c}\}$$

and

$$\bigoplus = \text{Set symmetric difference[Addition under} Z_2].$$

Combining Lemma 1 and 2, we conclude:

**Theorem 1** *Output of Algorithm 1 is* $\bigcup \partial\mathcal{R}(c_i)$, *where* $c_i \in \{$*Critical tetrahedra*$\}$ *and* $pers(c_i) \geq \delta$.

Figure 4 shows a 2D analogue of the merging procedure. The triangles $t_1$(red), $t_2$(blue) are critical triangles, and $e_1$(red) is a critical edge. We cancel the pair $(e_1, t_2)$ by reversing the V-path from $t_2$ to $e_1$.

Although we have used 2D analogue to explain the merging procedure, it is proven that for 2D case not all triangle–edge pairs are cancellable. In this paper, we are cancelling just the tetrahedron–triangle pairs arising from a triangulated 3-manifold, and it turns out that they are always cancellable [3].

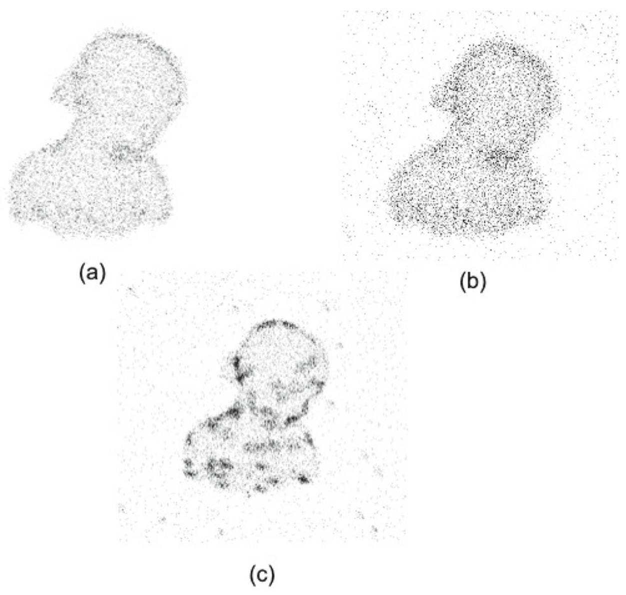

Fig. 5 Point-cloud of Bimba with three types of noise. (**a**) Individual points perturbed along vertex normal. (**b**) Outliers. (**c**) Randomly clustered noise

## 5 Applications

We are now ready to discuss the applications of Algorithm 1 described in the previous section. Our main focus will be on two such areas, namely denoising noisy point-clouds and extraction of surfaces from medical volumes where this algorithm shows resilience to noise.

### 5.1 Denoising noisy point-clouds

Suppose that we are given a noisy point-cloud $\mathcal{P} \subset \mathbb{R}^3$. We limit our discussion to the noise in $\mathcal{P}$ coming from *outliers, local-perturbations of the original points* along surface normal (tangential perturbations do not contribute to this type of noise) and *randomly clustered noise*. An example for each type of noise and its effect on 3D point-cloud sampled from a surface can be seen in Fig. 5.

#### 5.1.1 Preprocessing

Note that we are not yet ready to run Algorithm 1 for denoising purpose as it takes a simplicial complex $\mathcal{K}$ and an intensity function, $I$ defined on it. A 3D triangulation, $\mathcal{K}$ of $\mathcal{P}$ would be enough to fulfil the first requirement. Many software packages such as CGAL and Tetgen [37,39] are available open source for this purpose. We refine $\mathcal{K}$, by adding the circumcentre of each triangle $T \in \mathcal{K}$, and obtain $\mathcal{K}'$. The next step is to define an intensity function on $\mathcal{K}'$. To accomplish this, a Gaussian kernel is used, $I_G : Vert(\mathcal{K}') \to \mathbb{R}$ defined as

$I_G(\mathbf{x}) = \sum_i \exp\left(-\frac{\|\mathbf{x}-\mathbf{p}_i\|_2^2}{s^2}\right)$, where $\mathbf{x} \in Vert(\mathcal{K}')$ and $\mathbf{p}_i \in \mathcal{P}$. The spread of the Gaussian kernel is determined by the *variance* $s^2$. The idea behind is, $I$ should capture the features effectively meaning higher value of $I$ should capture higher density, with the assumption that higher density corresponds to samples from the surface and the lower density should correspond to noise. Note that to reduce variation in $\delta$, the persistence threshold parameter of Algorithm 2, we work with *normalized* $I_G$ instead of $I_G$. We would like to choose a suitable $s$ so that it covers the whole point-cloud while maintaining the density. With this in mind, we set

$$s = \frac{1}{|\mathcal{P}|}\sum_{\mathbf{p}_i\in\mathcal{P}} \min_{\mathbf{p}_j\in\mathcal{P}, i\neq j} \|\mathbf{p}_j - \mathbf{p}_i\|_2$$

where $|\mathcal{P}|$ is the number of points in the noisy point-set $\mathcal{P}$. In other words, we set $s$ to be the average nearest-neighbor Euclidean distance of the noisy point-cloud. We consider piecewise constant interpolation of $I_G$,

$$I(\sigma_i) = \max\{I(u_i) : u_i \in Vert(\sigma_i)\}$$

where $\sigma_i$ are the simplices of the simplicial complex $\mathcal{K}'$ making $I$ appropriate as an input to our region-growing algorithm.

Note that we collect only the vertices of the triangles output by Algorithm 1 since our domain is point-cloud.

### 5.1.2 Evaluation metrics

Denote the ground truth and the denoised point-cloud as $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{N_1}\}$ and $\mathcal{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{N_2}\}$, respectively, with $\mathbf{x}_i, \mathbf{y}_i \in \mathbb{R}^3$. We use the following metrics to measure how faithfully $\mathcal{Y}$ adheres to $\mathcal{X}$:

1. Mean square error (MSE): The MSE is defined as

$$\frac{1}{N_1}\sum_{\mathbf{x}_i\in\mathcal{X}} \min_{\mathbf{y}_j\in\mathcal{Y}} \|\mathbf{x}_i - \mathbf{y}_j\|_2^2 + \frac{1}{N_2}\sum_{\mathbf{y}_i\in\mathcal{Y}} \min_{\mathbf{x}_j\in\mathcal{X}} \|\mathbf{y}_i - \mathbf{x}_j\|_2^2 \quad (1)$$

2. Signal-to-noise ratio (SNR) in dB: SNR measured in dB is defined as

$$SNR = 10\log\frac{1/N_2 \sum_{\mathbf{y}_i\in\mathcal{Y}} \|\mathbf{y}_i\|_2^2}{MSE} \quad (2)$$

3. Mean absolute error (MAE): MAE is defined similarly as MSE, but $l_2$-norm is replaced with $l_1$.

$$MAE = \frac{1}{N_1}\sum_{\mathbf{x}_i\in\mathcal{X}} \min_{\mathbf{y}_j\in\mathcal{Y}} |\mathbf{x}_i - \mathbf{y}_j| + \frac{1}{N_2}\sum_{\mathbf{y}_i\in\mathcal{Y}} \min_{\mathbf{x}_j\in\mathcal{X}} |\mathbf{y}_i - \mathbf{x}_j| \quad (3)$$

4. Hausdorff distance: Hausdorff distance is defined as

$$d_H(\mathcal{X}, \mathcal{Y}) = max\left\{\sup_{\mathbf{x}\in\mathcal{X}}\inf_{\mathbf{y}\in\mathcal{Y}} d(\mathbf{x},\mathbf{y}), \sup_{\mathbf{y}\in\mathcal{Y}}\inf_{\mathbf{x}\in\mathcal{X}} d(\mathbf{x},\mathbf{y})\right\} \quad (4)$$

where $d(\mathbf{x}, \mathbf{y})$ is the Euclidean distance between $\mathbf{x}$ and $\mathbf{y}$.

### 5.1.3 Experiments

We now discuss the results of our algorithm on noisy point-clouds sampled from three different surface meshes: *Bimba*, *Botijo* and *Fertility* from the Aim@Shape repository [23].

The cardinality of each noise component can be found in Table 1. We would like to mention that choosing a suitable $\delta$ is tricky. For now, we choose $\delta$ that gives the best SNR (Sect. 5.1.2 for evaluation metrics) via a grid search (in the interval of [0.0001, 0.001] with a step of 0.0001).

### 5.1.4 Comparison with other methods

Since this is a proof of concept, it will be unrealistic to compare with non-TDA-based dedicated state-of-the-art methods such as *Dictionary Learning* [40] or supervised methods such as *PointCleanNet* [33]. Instead, we perform comparison with some of the commonly used algorithms like CGAL's outlier-removal [27], moving least squares (MLS) [1,30], weighted locally optimal projection (WLOP) [29] and *Parfree-Declutter*. As CGAL's outlier-removal deletes the points with large average squared distance to their nearest neighbors, it successfully clears the outliers but fails to remove the randomly clustered points (see Fig. 7 and Table 3).

Moreover, it deletes some of the points sampled from the actual surface without removing the clustered noise components if average_spacing (the primary parameter in the algorithm that determines the extent of outlier rejection) is decreased. We conclude this section by mentioning *Parfree-Declutter* produces comparable output but as compared to ours *Parfree-Declutter* is not based on discrete Morse theory. Table 4 shows the runtime of different algorithms. All of them were run in 1.6 GHz Dual-Core Intel Core i5 processor with 16 GB 2133 MHz memory. For fairness, all of the algorithms were in C++. Although MLS and Outlier-removal have less runtime compared to our algorithm, the results are less robust as seen in Fig. 7.

### 5.1.5 The parameter $\delta$

Intuitively $\delta$ can be thought of as a simplification parameter. A larger $\delta$ would simplify more, i.e., merge more regions, and a smaller $\delta$ would simplify less. As a result, high $\delta$ would remove triangles aggressively and lower $\delta$ would simplify less meaning a lot of redundant triangles and consequently

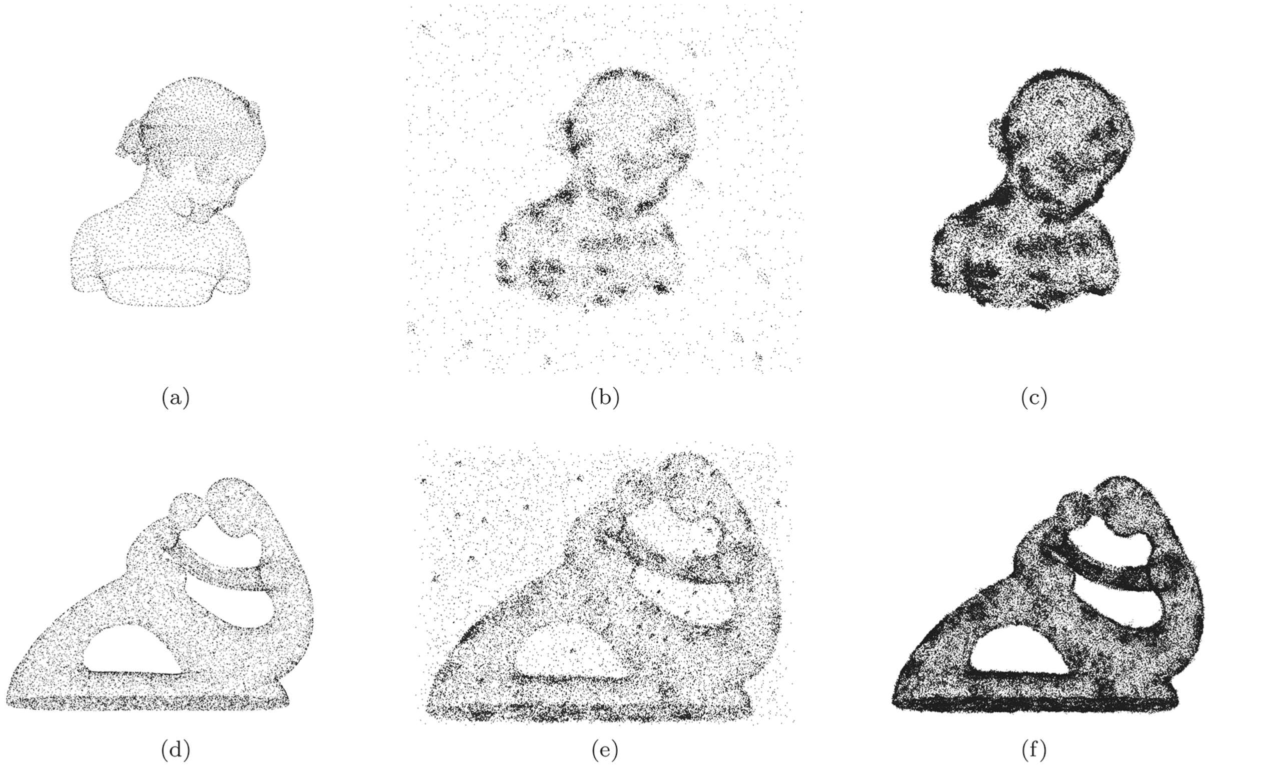

**Fig. 6** Denoising noisy point-clouds. In each row from left to right-ground truth, noisy input, vertices of the triangles jettisoned by Algorithm 1 is shown. The top row corresponds to Bimba and the last row is Fertility. Corresponding $\delta$s are $\delta_{\text{Bimba}} = 0.0001$, and $\delta_{\text{Fertility}} = 0.0005$

**Table 1** Amount of noise added in each dataset

| Dataset | # Outliers | # Perturbed points | # Randomly clustered noise | # Total vertices |
|---|---|---|---|---|
| Bimba | 3000 | 10,000 | 500 | 13,500 |
| Botijo | 3000 | 10,000 | 500 | 13,500 |
| Fertility | 5000 | 20,000 | 1000 | 26,000 |

**Table 2** Details of preprocessing

| Dataset | # Vertices before refinement | # Vertices after refinement | # Faces in triangulation | $s$ for Gaussian |
|---|---|---|---|---|
| Bimba | 13,500 | 184,432 | 170,932 | 0.03 |
| Botijo | 13,500 | 182,704 | 169,204 | 0.04 |
| Fertility | 26,000 | 329,348 | 355,348 | 24.48 |

redundant vertices in the output. Note that $\delta$ is a persistence threshold and since persistence homology is robust against small perturbations [12] our algorithm is not sensitive w.r.t. choice of $\delta$.

As shown in Fig. 8, SNR of the output remains constant at 75.4237 dB over the range of $\delta \in [0.0001, 0.001]$ for the model Bimba. For Botijo, SNR varies between $[74.20, 79.32]dB$. SNR for Fertility remains in the range $[84.98, 88.67]dB$ for $\delta \in [0.0001, 0.0005]$. For $\delta > 0.0005$, a large region merged and we see a large drop in SNR.

## 5.2 Extracting surfaces from noisy volumes

### 5.2.1 Problem setup

Suppose that we are given $\mathcal{M}$, a 2-manifold embedded in 3D grid $\mathcal{D} \subset \mathbb{R}^3$ and a function $g : \mathcal{D} \to \mathbb{R}$ that concentrates around $\mathcal{M}$. Our goal is to compute $\mathcal{M}'$ approximating $\mathcal{M}$. Note that with little adjustments Algorithm 2 works when the domain is a cubical complex.

**Table 3** MSE, SNR, MAE and Hausdorff distance between ground truth and denoised point-cloud of standard denoising methods. For Region Grow $\delta$s are $\delta_{\text{Bimba}} = 0.0001$, $\delta_{\text{Botijo}} = 0.0001$ and $\delta_{\text{Fertility}} = 0.0005$

| | Bimba | | | | Botijo | | | | Fertility | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | MSE | SNR | MAE | Hausdorff distance | MSE | SNR | MAE | Hausdorff distance | MSE | SNR | MAE | Hausdorff distance |
| Before Denoising | 0.1418 | 15.2708 | 0.2064 | 2.0334 | 0.1595 | 28.5620 | 0.2190 | 2.5931 | 15241.0837 | 47.9158 | 62.5225 | 1454.4524 |
| MLS | 0.0520 | 18.5160 | 0.0795 | 1.7902 | 0.0576 | 36.3074 | 0.1024 | 2.2151 | 3930.7625 | 60.5534 | 32.3896 | 1231.8108 |
| WLOP | 0.1274 | 13.0256 | 0.2929 | 1.9550 | 0.1484 | 28.4056 | 0.2718 | 2.5618 | 15265.9950 | 47.8995 | 63.5011 | 1454.4524 |
| Outlier Removal | 0.0500 | 18.7505 | 0.0815 | 1.7899 | 0.0470 | 38.1255 | 0.0857 | 2.2147 | 3580.0231 | 61.5189 | 31.4066 | 1231.8538 |
| Parfree-Declutter | 0.0001 | 71.0202 | 0.0142 | 0.1399 | 0.0009 | 76.1071 | **0.0296** | 0.4216 | 1542.1124 | 70.0517 | 29.4396 | 486.0903 |
| Region Grow | **8.334e-05** | **75.4237** | **0.0113** | **0.0513** | **0.0007** | **78.3814** | 0.0323 | **0.1623** | **230.7473** | **88.6660** | **19.2538** | **276.9980** |

Bold values indicate the minimum error achieved among the performed experiments

**Table 4** Running time (in seconds) of different denoising methods on different datasets. $\infty$ denotes the program took greater than 30 minutes to terminate

| Name | Bimba | Botijo | Fertility |
|---|---|---|---|
| MLS | 0.169 | 0.086 | 0.816 |
| WLOP | 682.489 | 101.426 | $\infty$ |
| Outlier Removal | 0.671 | 0.647 | 1.318 |
| Parfree-Declutter | $\infty$ | $\infty$ | $\infty$ |
| Region Grow | 57.538 | 57.847 | 123.3034 |

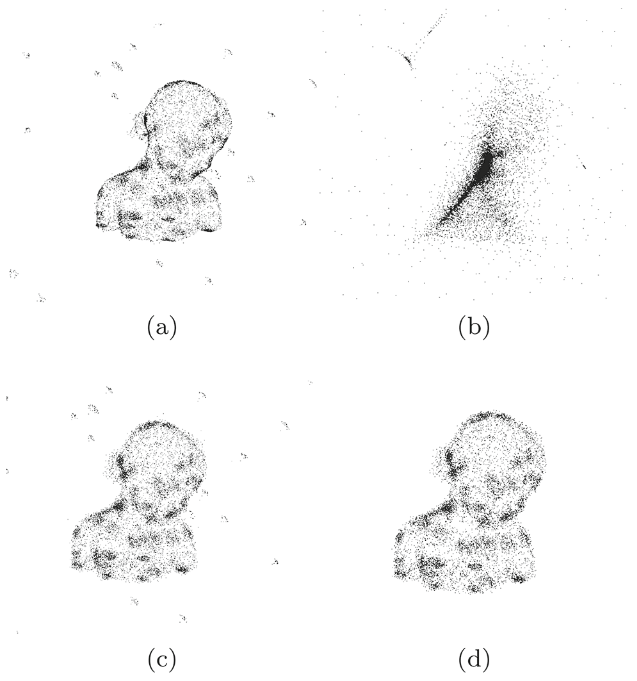

**Fig. 7** (Output of different denoising algorithms. (**a**) Output of MLS. (**b**) Output of WLOP. (**c**) Output of CGAL's outlier removal. (d) Output of Parfree-Declutter. Parfree-Declutter is the only algorithm that was able to denoise the randomly clustered noise

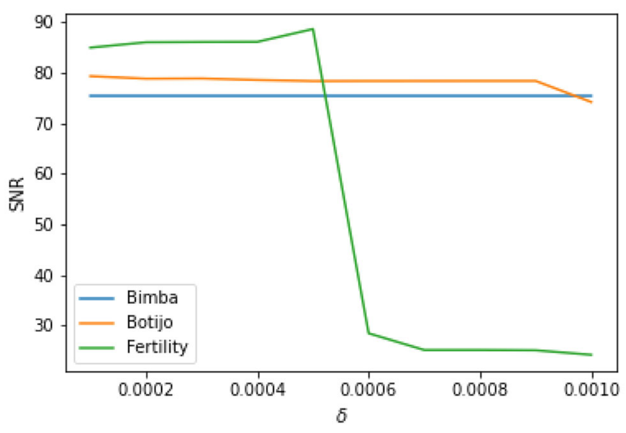

**Fig. 8** SNR of different models w.r.t. $\delta$

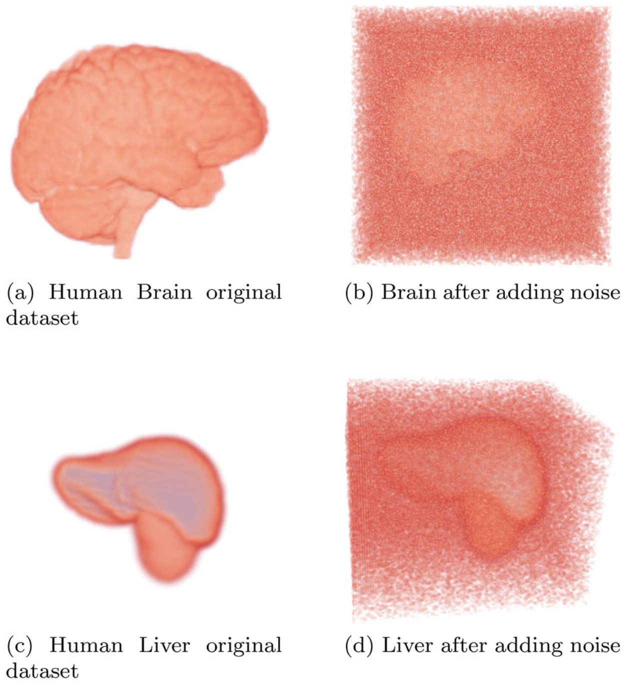
(a) Human Brain original dataset
(b) Brain after adding noise
(c) Human Liver original dataset
(d) Liver after adding noise

**Fig. 9** Original datasets and noisy datasets

**Table 5** Dimension of datasets. Noise% is the number of voxel values replaced per 100 voxels

| Dataset | Dimension | Noise% |
|---|---|---|
| INRIA Liver | 100 x 100 x 100 | 30% |
| Human Brain | 96 x 128 x 128 | 15% |

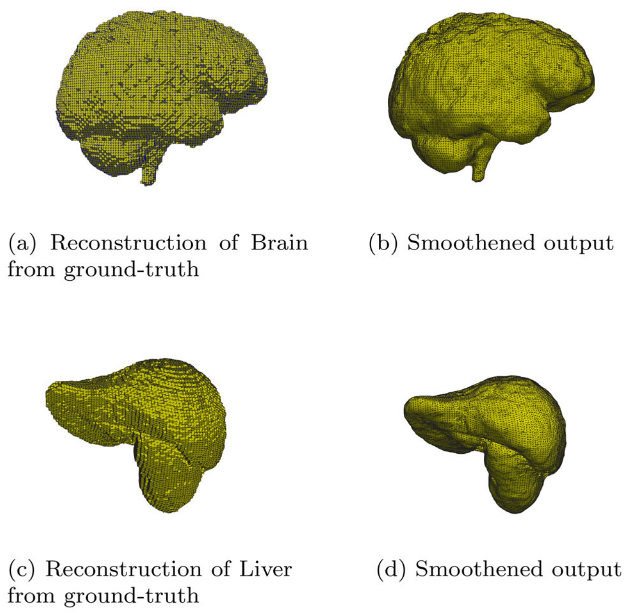
(a) Reconstruction of Brain from ground-truth
(b) Smoothened output
(c) Reconstruction of Liver from ground-truth
(d) Smoothened output

**Fig. 10** Surface extraction from volumes. Top row is the human brain and the bottom one is human liver. The left column is the raw output of Algorithm 1, and the right column is the smoothened version of the same

### 5.2.2 Preprocessing

Similar to previous subsection, we begin by triangulating $\mathcal{D}$ and obtaining the simplicial complex $\mathcal{K}$ upon which Algorithm 1 will act. Unlike previous example, the intensity function is already available and we will use $I = -|\nabla g|$. Intuitively, near the surface $|\nabla g|$ is high (after inversion low) and we wish to place our region boundaries at that place exactly so that it approximates $\mathcal{M}$ effectively. Notice that the regions correspond to critical maxima and taking -$|\nabla g|$ instead of $|\nabla g|$ makes the noisy sites act as minima and thus are not captured by our algorithm.

### 5.2.3 Post-processing

Because of the noise present in the image, there may be spurious maxima (although we have taken -$|\nabla g|$, but points in the neighborhood of noisy sites can act as maxima) with high persistence. They will act as a root of region, and boundary of such regions will be subset of $\mathcal{E}$, output of our algorithm. Evidently $\mathcal{E} = \widetilde{\mathcal{M}} \bigcup n$, where $\widetilde{\mathcal{M}}$ are the surface approximating $\mathcal{M}$ and $n$ is the boundary of spurious regions present due to noise. We can remove $n$ by taking larger components, say $x$ percentage of components, of $\mathcal{E}$. By larger we mean in terms of the radius of enclosed region or no. of triangles in that component.

### 5.2.4 Experiments

In this section, we test our algorithm on two volume data. Namely, the INRIA liver and the Human Brain.
*Noise Addition:* We introduce noise to the image $\mathcal{D}$ by replacing intensity value at random locations with a random value chosen uniformly within the range of minimum to maximum intensity (see Fig. 9). Details for each dataset are given in Table 5.

### 5.2.5 Results

Figure 10 shows the results of our algorithm on the ground truth. $\delta$s corresponding to the reconstruction are $\delta_{brain} = 20$ and $\delta_{liver} = 3$.

Notice the presence of spurious regions outside of the reconstructed mesh in Fig. 11a, b. Also the triangles jettisoned by Algorithm 1 are a subset of the original triangulated 3D grid. It is not uncommon in these cases to post-process and smoothen the resulting mesh. To remove the spurious regions, we used the strategy mentioned in the post-processing paragraph of Sect. 5.2. To smooth the surfaces, we used Par-

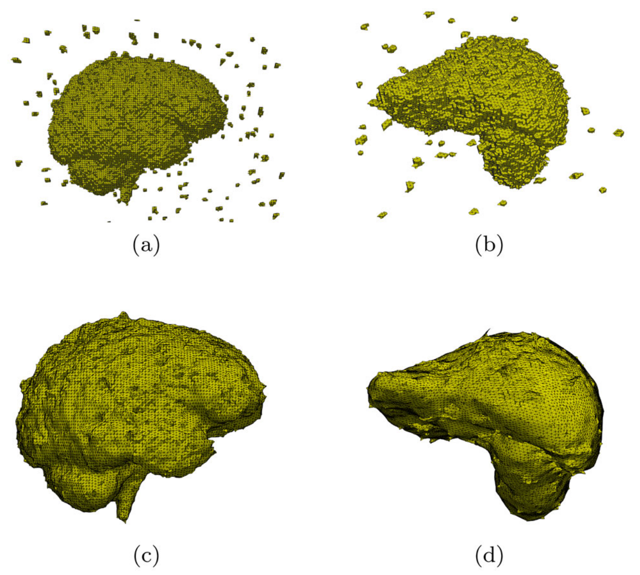

**Fig. 11** Output of Algorithm 1. (**a**, **b**) Triangles output by our algorithm on human brain($\delta = 20$) and liver dataset($\delta = 3$). (**c**, **d**) after post-processing of the same

aview's smoothing filter. Figure 11c and d shows the result after post-processing and smoothing.

### 5.2.6 Comparison with baseline

A common yet simple approach to denoise 3D datasets is *thresholding*. However due to non-homogeneity of the datasets that were used by us, a single threshold $t$ fails to denoise the image. *Isosurface detection* meets the same fate as choosing an iso-value $i$ is hard. As smaller value of $i$ produces redundant noisy features, whereas larger value of $i$ destroys features of data. We also compare our results with CGAL's 3D surface mesher [34]. Since CGAL's 3D mesher requires a grey-level set, it fails to reconstruct the surface as seen in Fig. 12e. Note that our algorithm bears resemblance with strategies used for simplification of merge and split trees [9,10,32] since they both have their origin rooted at Morse theory and employ similar techniques for simplification of trees. Although persistence threshold can be used as a simplification parameter of contour trees [6,42], our goal and scenario are entirely different. Algorithm 1 operates on the dual graph with tetrahedra as nodes and triangles as edges, ergo bypassing the need for computation of contour tree. Also, note that, our end goal is not to simplify the dual graph, but to argue that this simple region-growing algorithm can denoise regular volumes. We do not have any theoretical guarantee yet, empirically our discrete Morse theoretic framework produces better results for such non-homogeneous data.

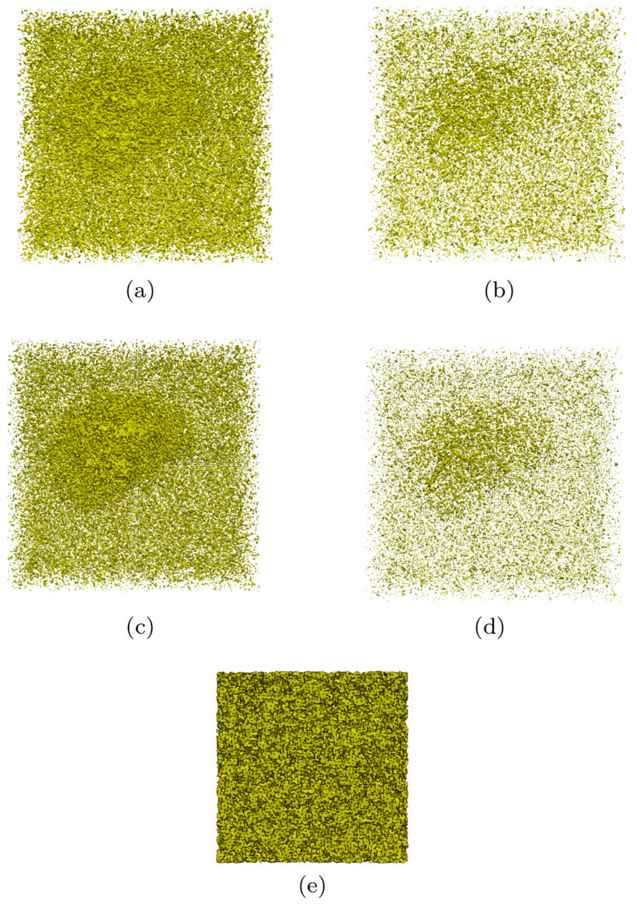

**Fig. 12** Results of conventional algorithms on human brain dataset. (**a**, **b**) Lower threshold value ($t = 600$) produces redundant features, while higher values ($t = 800$) destroy important ones. (**c**, **d**)Isosurface detection fails to reconstruct the surface. (**e**) Result of CGAL's 3$D$ surface mesher

## 6 Conclusion

In this paper, we have presented a robust framework for denoising point-clouds and 3D digital images. While our algorithm is empirically very good, we currently lack a theoretical guarantee. In our view, the major remaining step is to provide a theoretical guarantee on the effectiveness of our algorithm. The result in [18] could provide such a roadmap. Finally, we encourage the reader to consult our full implementation at https://github.com/soham0209/Denoise-Discrete-Morse.

**Acknowledgements** The author would like to acknowledge Prof. Tamal K. Dey and Prof. Yusu Wang for their guidance, helpful discussions on persistent homology and discrete Morse theory. The author would also like to thank Ryan Šlechta for his detailed suggestions and thoughtful comments on this manuscript.

## References

1. Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., Silva, C.T.: Point set surfaces. Visualization **2001**, 21–28 (2001)
2. Attali, D., Glisse, M., Hornus, S., Lazarus, F., Morozov, D.: Persistence-sensitive simplification of functions on surfaces in linear time. Presented at the workshop TopoInVis'09 **9**, 23–24 (2009)
3. Bauer, U., Lange, C., Wardetzky, M.: Optimal topological simplification of discrete functions on surfaces. Discr. Comput. Geom. **47**(2), 347–377 (2012)
4. Beksi, W.: Topological Methods for 3D Point Cloud Processing. Ph.D. thesis, Retrieved from the University of Minnesota digital conservancy (2018). https://hdl.handle.net/11299/201078
5. Berger, M., Tagliasacchi, A., Seversky, L.M., Alliez, P., Guennebaud, G., Levine, J.A., Sharf, A., Silva, C.T.: A survey of surface reconstruction from point clouds. In: Computer Graphics Forum, vol. 36, pp. 301–329. Wiley Online Library (2017)
6. Bock, A., Doraiswamy, H., Summers, A., Silva, C.: Topoangler: interactive topology-based extraction of fishes. IEEE Trans. Visual. Comput. Graph. **24**(1), 812–821 (2018). https://doi.org/10.1109/TVCG.2017.2743980
7. Bremer, P.T., Hamann, B., Edelsbrunner, H., Pascucci, V.: A topological hierarchy for functions on triangulated surfaces. IEEE Trans. Visual. Comput. Graph. **10**(4), 385–396 (2004)
8. Buchet, M., Dey, T.K., Wang, J., Wang, Y.: Declutter and resample: towards parameter free denoising. In: 33rd international symposium on computational geometry. SoCG 2017, pp. 231–2316. Schloss Dagstuhl, Leibniz-Zentrum fü Informatik GmbH (2017)
9. Carr, H., Snoeyink, J., Axen, U.: Computing contour trees in all dimensions. Comput. Geom. **24**(2), 75–94 (2003)
10. Carr, H., Snoeyink, J., van de Panne, M.: Contour tree simplification with local geometric measures. In: 14th annual fall workshop on computational geometry, vol. 80, p. 51. Citeseer (2004)
11. Chazal, F., Guibas, L.J., Oudot, S.Y., Skraba, P.: Analysis of scalar fields over point cloud data. In: Proc. 20th ACM-SIAM Sympos. Discrete Algorithms, pp. 1021–1030 (2009)
12. Cohen-Steiner, D., Edelsbrunner, H., Harer, J.: Stability of persistence diagrams. Discr. Comput. Geom. **37**(1), 103–120 (2007). https://doi.org/10.1007/s00454-006-1276-5
13. De Floriani, L., Fugacci, U., Iuricich, F., Magillo, P.: Morse complexes for shape segmentation and homological analysis: discrete models and algorithms. In: Computer Graphics Forum, vol. 34, pp. 761–785. Wiley Online Library (2015)
14. Delfinado, C.J.A., Edelsbrunner, H.: An incremental algorithm for betti numbers of simplicial complexes. In: Proceedings of the ninth annual symposium on computational geometry, SCG '93, p. 232–239. Association for computing machinery, New York, NY, USA (1993). https://doi.org/10.1145/160985.161140
15. Delgado-Friedrichs, O., Robins, V., Sheppard, A.: Skeletonization and partitioning of digital images using discrete morse theory. IEEE Trans. Pattern Anal. Machine Intell. **37**(3), 654–666 (2015). https://doi.org/10.1109/TPAMI.2014.2346172
16. Dey, T.K., Li, K., Luo, C., Ranjan, P., Safa, I., Wang, Y.: Persistent heat signature for pose-oblivious matching of incomplete models. Comput. Graph. Forum (2010)
17. Dey, T.K., Slechta, R.: Edge contraction in persistence-generated discrete morse vector fields. Comput. Graph. **74**, 33–43 (2018). https://doi.org/10.1016/j.cag.2018.05.002
18. Dey, T.K., Wang, J., Wang, Y.: Graph Reconstruction by Discrete Morse Theory. In: B. Speckmann, C.D. Tóth (eds.) 34th International Symposium on Computational Geometry (SoCG 2018), Leibniz International Proceedings in Informatics (LIPIcs), vol. 99, pp. 31:1–31:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2018). https://doi.org/10.4230/LIPIcs.SoCG.2018.31. http://drops.dagstuhl.de/opus/volltexte/2018/8744
19. Dey, T.K., Wang, J., Wang, Y.: Graph reconstruction by discrete morse theory. CoRR (2018). arXiv:1803.05093
20. Dey, T.K., Wang, Y.: Computational topology for data analysis (2021). https://www.cs.purdue.edu/homes/tamaldey/book/CTDAbook/CTDAbook.pdf
21. Edelsbrunner, H., Harer, J.: Computational topology: an introduction. Am. Math. Soc. (2010)
22. Edelsbrunner, H., Letscher, D., Zomorodian, A.: Topological persistence and simplification. In: Proceedings 41st annual symposium on foundations of computer science, pp. 454–463 (2000). https://doi.org/10.1109/SFCS.2000.892133
23. Falcidieno, B.: Aim@ shape project presentation. In: Proceedings Shape Modeling Applications, 2004., p. 329. IEEE (2004)
24. Forman, R.: A discrete morse theory for cell complexes. In: in Geometry, Topology 6 Physics for Raoul Bott. Citeseer (1995)
25. Forman, R.: Morse theory for cell complexes. Adv. Math. **134**(1), 90–145 (1998)
26. Forman, R.: A user's guide to discrete morse theory. Sém. Lothar. Combin **48**, 35pp (2002)
27. Giraudot, S.: 3D point set. In: CGAL User and Reference Manual, 5.1 edn. CGAL Editorial Board (2020). https://doc.cgal.org/5.1/Manual/packages.html#PkgPointSet3
28. Hodge, V.J., Austin, J.: A survey of outlier detection methodologies. Artif. Intell. Rev. **22**(2), 85–126 (2004). https://doi.org/10.1007/s10462-004-4304-y
29. Huang, H., Li, D., Zhang, H., Ascher, U., Cohen-Or, D.: Consolidation of unorganized point clouds for surface reconstruction. ACM Trans. Graph. **28**(5), 1–7 (2009). https://doi.org/10.1145/1618452.1618522
30. Levin, D.: The approximation power of moving least-squares. Math. Comput. **67**(224), 1517–1531 (1998)
31. Munkres, J.R.: Elements of Algebraic Topology. CRC Press, Boca Raton (2018)
32. Pascucci, V., Cole-McLaughlin, K., Scorzelli, G.: Multi-resolution computation and presentation of contour trees. In: Proc. IASTED conference on visualization, imaging, and image processing, pp. 452–290. Citeseer (2004)
33. Rakotosaona, M.J., La Barbera, V., Guerrero, P., Mitra, N.J., Ovsjanikov, M.: Pointcleannet: Learning to denoise and remove outliers from dense point clouds. In: Computer Graphics Forum, vol. 39, pp. 185–203. Wiley Online Library (2020)
34. Rineau, L., Yvinec, M.: 3D surface mesh generation. In: CGAL User and Reference Manual, 5.2 edn. CGAL Editorial Board (2020). https://doc.cgal.org/5.2/Manual/packages.html#PkgSurfaceMesher3
35. Shinagawa, Y., Kunii, T.L.: Constructing a reeb graph automatically from cross sections. IEEE Ann. Hist. Comput. **11**(06), 44–51 (1991)
36. Shinagawa, Y., Kunii, T.L., Kergosien, Y.L.: Surface coding based on morse theory. IEEE Comput. Graph. Appl. **11**(5), 66–78 (1991)
37. Si, H.: Tetgen, a delaunay-based quality tetrahedral mesh generator. ACM Trans. Math. Softw. (TOMS) **41**(2), 1–36 (2015)
38. Sousbie, T.: The persistent cosmic web and its filamentary structure–i. theory and implementation. Monthly notices of the royal astronomical society **414**(1), 350–383 (2011)
39. The CGAL Project: CGAL User and Reference Manual, 5.0.2 edn. CGAL Editorial Board (2020). https://doc.cgal.org/5.0.2/Manual/packages.html
40. Xiong, S., Zhang, J., Zheng, J., Cai, J., Liu, L.: Robust surface reconstruction via dictionary learning. ACM Trans. Graph. (TOG) **33**(6), 1–12 (2014)
41. Zhang, J.: Advancements of outlier detection: a survey. ICST Trans. Scalable Inform. Syst. **13**(1), 1–26 (2013)

42. Zhou, J., Xiao, C., Takatsuka, M.: A multi-dimensional importance metric for contour tree simplification. J. Visual. **16**(4), 341–349 (2013)

**S. Mukherjee** is a Graduate Student of Purdue University and is currently doing his PhD in Computer Science. Soham joined Department of Computer Science, Purdue University in Fall 2020. His research focuses on Computational Topology and Topological Data Analysis. He received his Bachelors in Electronics and Telecommunication Engineering from Jadavpur University, Kolkata.