HAL open science

Persistent Homology Computation Using Combinatorial Map Simplification

Guillaume Damiand, Rocio Gonzalez-Diaz

To cite this version:

Guillaume Damiand, Rocio Gonzalez-Diaz. Persistent Homology Computation Using Combinatorial Map Simplification. International Workshop on Computational Topology in Image Context, Jan 2019, Malaga, Spain. pp.26-39, 10.1007/978-3-030-10828-1_3 . hal-02002963

HAL Id: hal-02002963

https://hal.science/hal-02002963v1

Submitted on 1 Feb 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Persistent Homology Computation Using Combinatorial Map Simplification

Guillaume Damiand and Rocio Gonzalez-Diaz

[1] Univ. Lyon, CNRS, LIRIS, UMR5205, F-69622 France
guillaume.damiand@liris.cnrs.fr
https://liris.cnrs.fr/guillaume.damiand/

[2] Universidad de Sevilla, Dpto. de Matemática Aplicada I, S-41012, Spain
rogodi@us.es
http://personal.us.es/rogodi/

Abstract. We propose an algorithm for persistence homology computation of orientable 2-dimensional (2D) manifolds with or without boundary (*meshes*) represented by 2D combinatorial maps. Having as an input a real function h on the vertices of the mesh, we first compute persistent homology of filtrations obtained by adding cells incident to each vertex of the mesh, The cells to add are controlled by both the function h and a parameter δ. The parameter δ is used to control the number of cells added to each level of the filtration. Bigger δ produces less levels in the filtration and consequently more cells in each level. We then simplify each level (cluster) by merging faces of the same cluster. Our experiments demonstrate that our method allows fast computation of persistent homology of big meshes and it is persistent-homology aware in the sense that persistent homology does not change in the simplification process when fixing δ.

Keywords: Persistent homology computation; 2D combinatorial map; mesh simplification

1 Introduction

Topological data analysis (TDA) is a relatively new field in computer science. One of the most useful concept in TDA is the one of persistent homology which is an algebraic method for measuring topological features (connected components, voids, cavities, etc) of shapes and functions. Two of the crucial ingredients of persistence are: (1) a *cell complex* to structure the data; and (2) a *filtration* which is a nested sequence of subcomplexes that starts with the empty complex and ends with the whole complex. See [1, 2] for initial reports and [3, 4] for a modern exposition of the field.

In [5], the authors proposed an efficient algorithm that computes persistent homology for 3D gray-scale images using the Morse-Smale complex previously obtained, which is much smaller than the input data, but with all necessary information. The authors first computed a combinatorial gradient vector field

(GVF) by a process presented in [6]. To do this, the cell complex is decomposed into the lower star of its vertices. The authors then computed persistent homology from the boundary map of the chain complex associated to the Morse-Smale complex induced by GVF.

In [7], we proposed an efficient algorithm for computing the homology of meshes (orientable manifolds with or without boundary), represented by 2D combinatorial maps (which are models of representation of meshes composed by vertices, edges linking two vertices, and 2D faces bounded by a closed path of edges), avoiding the time-consuming step of constructing and modifying boundaries and coboundaries of cells. The process consists of merging faces if they share a common edge, guaranteeing that the structure of combinatorial map and the homology information of the mesh is preserved until the end of the process.

In this paper we extend our work to compute persistent homology of meshes. First, as in [7], a simplification process is made to improve computation time. Now, faces as dispatched in clusters depending on a parameter δ and only faces of the same cluster are merged. For constructing the cluster the following rule is used: two faces are in a same cluster if there is a path of vertices of these two faces of length smaller than δ. At the end of the process, a smaller than the input 2D combinatorial map is obtained. To obtain persistent homology of the simplified mesh, lower-start filtration induced by a function h on its vertices (in our case, h is the height function) is computed. Varying the parameter δ, the filtration varies and also its persistent homology.

The paper is organized as follows. Section 2 recalls the background of the paper regarding combinatorial maps and persistent homology. Section 3 is the main section of the paper and presents our method to compute persistent homology starting from a particular filtration constructed from the height function and a parameter δ. Several experimental and computational results are presented in Section 4. Finally, we summarize the paper with a brief discussion about future work in Section 5.

2 Preliminary Notions

In this section we recall the needed background of the paper regarding combinatorial maps and persistent homology.

2.1 2D combinatorial maps

A 2D combinatorial map [8, 9], called 2-map, is a model of representation of a mesh, which is composed by *i-cells*: *vertices* or 0-cells associated with points, *edges* or 1-cells which link two vertices, and *faces* or 2-cells which are bounded by a closed path.

Two cells are *incident* if one cell belongs to the boundary of the other one; while two i-cells c_1 and c_2 are *adjacent* if it exists one $(i-1)$-cell incident to both c_1 and c_2. An edge e is *dangling* if it is incident to one vertex v such that no other edge than e is incident to v. An edge is *isolated* if it has no adjacent edge.

An edge incident to two different faces is called *inner*. Such an edge is necessarily not dangling nor isolated. Lastly, an edge is called *border* if it is incident to only one face and if it touches the boundary of the mesh. See Figure 1(a).

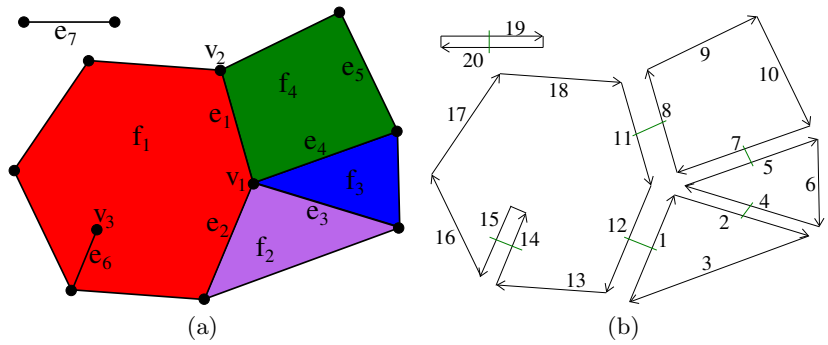


Fig. 1. (a) Example of a mesh having 5 faces (the four faces incident to vertex v_1, and the "degenerated one" bounded twice by edge e_7), 14 edges (e_6 is dangling, e_7 is isolated, $\{e_1, e_2, e_3, e_4\}$ are inner and the rest are border) and 12 vertices. (b) The corresponding 2-map has 20 darts. Images taken from [7].

The different elements of a mesh are encoded in a 2-map by *darts* and two mappings between these dart: β_1 and β_2:
$\boldsymbol{\beta_2}$: A dart is an orientation of an edge. If an edge separates two faces, it is described by two darts d_1, d_2 in the 2-map linked by β_2 (i.e., $\beta_2(d_1) = d_2$ and $\beta_2(d_2) = d_1$). These two darts represent the two possible orientations of the edge (for example $\beta_2(8) = 11$ and $\beta_2(11) = 8$ in Figure 1(b)). Each border edge is described by only one dart d in the 2-map, linked by β_2 with a special element $\varnothing$ (cf. for example dart 10 in Figure 1(b) which describes border edge e_5).$\boldsymbol{\beta_1}$: For each dart d, $\beta_1(d)$ is the dart following dart d and belonging to the same face than d (for example $\beta_1(1) = 2$ in Figure 1(b)). Note that a 2-map is oriented and thus described a given orientation of the mesh.

A dart belongs exactly to one vertex, one edge and one face, and thus each cell of the mesh is described by a set of darts in the 2-map. For example, in Figure 1(b), vertex v_1 is described by the set of darts $\{2, 5, 8, 12\}$. Note that this is a very important property of 2-map. Even an isolated edge (like e_7 in Figure 1(a)) belongs to one face (which explain why we have 5 faces and not 4 in Figure 1(a)).

The different type of edges can be detected in a 2-map thanks to particular configurations of darts and β links (for example an edge is isolated if $\beta_1(\beta_1(d)) = d$, d being of of the dart of the edge).

The algorithm presented in this paper for computing persistent homology on meshes used a modified version of Algorithm 1 detailed below which was presented in [7] to compute the minimal 2-map (i.e. with minimal number of

cells) describing a given mesh. The algorithm uses two operations on 2-maps: *edge removal* and *edge contraction*. It simplifies a given combinatorial map in its minimal form while preserving all the homology information. The proof that Algorithm 1 preserves homology information is given in [10].

Algorithm 1: Simplification of a mesh (modified version of Algorithm 1 of [7]).

Input: A 2-map M representing the mesh.
Output: The simplified 2-map corresponding to M.

```
foreach edge e of M do
    if e is an inner edge then remove e;
foreach edge e of M do
    while e is dangling do
        e' ← one edge adjacent to e;
        remove e;   e ← e';
foreach edge e of M do
    if e is not a loop then
        contract e;
```

2.2 Persistent Homology

In this subsection we give elementary notions from topology needed to understand the rest of the paper. In particular, we introduce the notion of homology and persistent homology. Precise definitions of homology can be found for example in [11], and definition of persistent homology for example in [4].

Homology can be thought as a method for defining k-dimensional holes (connected components, tunnels, voids) in a given mesh. For example. a 1-cycle is a closed path and a 1-boundary is the boundary of a 2D manifold. Then, 1-homology classes (which represent tunnels) are equivalence classes of 1-cycles modulo 1-boundaries. This concept can be generalized to k-homology classes. Finally, k-homology groups are the groups of k-homology classes.

Persistent homology captures the topological changes occurring in a growing sequence of meshes, called *filtration*. During the growth of a mesh, homology classes of different dimension may appear (be born) and disappear (die). Filtrations are frequently constructed using a real-valued function h on the vertices of the mesh M. For example, the *lower-start filtration* is computed as follows:

- First, order the vertices of M in a non-decreasing way,

$$h(v_1) \leq h(v_2) \leq \cdots \leq h(v_n).$$

- Second, compute the lower-star of a vertex v in M, which is the set of cells of M incident to v whose vertices all have function values at most $h(v)$.

- Define M_i as the union of the lower-star of all vertices of M whose function value is at most $h(v_i)$.
  This way, if $h(v_{i-1}) < h(v_i)$ then $M_i \setminus M_{i-1}$ is the set of cells of M having a vertex with function value exactly $h(v_i)$.
  And if $h(v_{i-1}) = h(v_i)$ then $M_{i-1} = M_i$.

The lower-star filtration of the mesh M induced by the function h is the sequence of nested meshes:

$$\emptyset = M_0 \subseteq M_1 \subseteq M_2 \subseteq \cdots \subseteq M_{n-1} \subseteq M_n = M.$$

Intuitively, imagine we sweep the mesh M in increasing values of the function h. At any real-value α, we consider the set of cells whose function value on their vertices is below or equal to α. As α increases, this gives us a sequence of subsets of M, growing larger and larger.

The topological evolution along the filtration is expressed by the corresponding sequence of homology groups. When adding the cells in order according to the filtration, new homology classes may born and some of them may later die when they become trivial or merge with another class. If a homology class γ is born at M_i and dies entering M_j then $h(v_j) - h(v_i)$ is the *persistence* of γ. If γ is born at M_i but never dies then its persistence is set to infinity. Homology classes with low persistence are considered noise and the ones that persist are considered features of the mesh.

The information obtained when computing persistent homology can be visualized as a *persistence barcode* which consists of the set of (birth, death) intervals, each interval recording a persistent homology event. The bottleneck distance is used to compare two persistence barcodes corresponding to two different filtrations of the same mesh. Given a bijection η between two persistence barcodes, we take the supremum L_∞-distance[3] between matched points and define the bottleneck distance by taking the infimum over all supremums.

In order to compute persistent homology, in this paper we have implemented a simplified version of the incremental algorithm for computing AT-models given in [12]. Given an ordering of the cells of the mesh, Algorithm 2 computes a triplet (M, H, f) where:

- M is the given mesh (decomposed in cells obtained from the combinatorial map). If σ is a k-cell, then $\partial(\sigma)$ is the set of $(k-1)$-cells in its boundary.
- H is a subset of cells of M called *surviving cells*. Fixed k, the set of all the surviving k-cells together with the addition operation $+$ (here $+$ means the disjoint union of sets) form the group $C_k(H)$ which is isomorphic to the k-dimensional homology group Hk of M.
- $f : C(M) \to C(H)$ maps each k-cell in M to a sum of surviving cells, satisfying that if $a, b \in C_k(M)$ are two homologous k-cycles then $f_k(a) = f_k(b)$. Let M_{σ_i} be the set of cells $\{\sigma_1, \ldots, \sigma_i\}$. Then, in the ith step of

[3] The L_∞-distance between points $u = (u_1, u_2)$ and $v = (v_1, v_2)$ in the extended plane is $\max\{|u_1 - v_1|, |u_2 - v_2|\}$,

Algorithm 2: Computing persistent homology (Algorithm 2 of [12]).

Input: An ordering of the cells of M: $\{\sigma_1, \ldots, \sigma_m\}$.
Output: Persistent homology.

Initialize $H := \emptyset$ and $f(\sigma_i) := 0$, for $1 \leq i \leq m$.
for $i = 1$ **to** m **do**
  if $f\partial(\sigma_i) = 0$ **then**
    $f(\sigma_i) := \sigma_i$, $H := H \cup \{\sigma_i\}$ (a new homology class was born).
  if $f\partial(\sigma_i) \neq 0$ **then**
    Let $\sigma_j \in f\partial(\sigma_i)$ s t. $j = \max\{\ \text{index}(\mu) : \mu \in f\partial(\sigma_i)\ \}$
    $H := H \setminus \{\sigma_j\}$ (an old homology class died).
    foreach $x \in M$ *such that* $\sigma_j \in f(x)$ **do**
      $f(x) := f(x) + f\partial(\sigma_i)$.

Algorithm 2, σ_i belongs to a k-cycle c in $C(M_{\sigma_i})$ if and only if $f\partial(\sigma_i) = 0$. This is why if $f\partial(\sigma_i) = 0$ then a new homology class was born (the one represented by the k-cycle c) and σ_i enters H. Otherwise, if $f\partial(\sigma_i) \neq 0$, then a homology class died, which is equivalent to say that an element of $f\partial(\sigma) \subseteq H$ is removed from H. The element to be removed from H will be the youngest one: $\max\{\ \text{index}(\mu) : \mu \in f\partial(\sigma_i)\ \}$, being $\text{index}(\mu)$ the position of the cell μ in the given ordered list of cells $\{\sigma_1, \ldots, \sigma_m\}$.

In [13] the authors establish a correspondence between the incremental algorithm for computing AT-models given in [12] and the one for computing persistent homology [4]. Since we are only interested in computing the persistence events, we only compute the set H and the map f. See Algorithm 2.

3 Computing Persistence

Our starting point is a subdivision of a mesh M (with or without boundaries) into vertices, edges and faces, and a real-valued function h on the vertices of the mesh.

Our method is based on three steps:

1. Simplification of the 2-map according to a parameter δ;
2. Computation of the lower-star filtration of the simplified mesh;
3. Computation of persistent homology of the given filtration.

Our goal in step 1 is to simplify the 2-map decreasing the number of faces in each level of the filtration in order to improve the computation time in Step 3 which is the more time-consuming step. Observe that persistent homology varies when δ varies since the filtration computed is different. Nevertheless, we have observe in the experiments that our simplification can be seen as a filtering of small persistent homology events.

3.1 2-Map Simplification

In this step, the 2-map is simplified by dispatching the faces into clusters and applying Algorithm 1 with constraints.

First, faces are dispatched into clusters according to the parameter δ. To compute such clusters, vertices of the mesh are ordered in a non-decreasing way by their height values $h(v)$. We assign a height value to each face with is the maximum value of the height of its vertices.

Then in the first cluster we add the first face f in the ordering and all the faces "at distance" less than δ. which means that there exists a path of vertices of these two faces of length smaller than δ. For example, if $\delta = 0$, only one face per cluster is added. If $\delta = 1$ all the faces sharing an edge with f are added. For any $\delta > 1$ all the faces at distance less than or equal to δ to f are added to the cluster. We repeat the process with the next face provided by the ordering that was not included in any cluster. We repeat the process until all faces are in a cluster.

After dispatching the faces in clusters, we apply Algorithm 1 with the following constraints:

- Faces merge (i.e, the inner shared edge e is removed) only if they belong to the same cluster.
- Besides, contrary to Algorithm 1, critical edges (separating faces belonging to two different clusters) are not removed here. Merging faces belonging to two different clusters could lead to loose a persistent event, and this is why we do not merge such faces.
- We do not use the contraction step (last **foreach** in Algorithm 1). Indeed, the simplified 2-map obtained here has several faces, contrary to Algorithm 1 computed without constraints that always produces one face per connected component. For this reason, the number of possible edges to contract is here smaller and thus we have observed no gain (and even sometimes a loss) when using the contraction step comparing to not use it.

3.2 Filtration

The second step in our algorithm for computing persistent homology is to compute the lower-star filtration (see Section 2.2) of the simplified mesh SM.

Observed that increasing the value of δ in Step 1 will decrease the different number of SM_j sets (i.e., the number of levels in the filtration), which increases the average number of cells belonging to a same SM_j, as illustrated in Figure 3 for the Neptune mesh and three different δ values. Note that bigger δ increases the number of simplifications done and thus decreases the size of the simplified combinatorial map. In this case, the persistent homology computed is not the same than the one obtained by the lower-star filtration on the original mesh (they only coincides when $\delta = 0$). Nevertheless, we have seen in our tests that the effect of the parameter $\delta > 0$ is to remove small persistent homology events. However this new possibility gives to users a way to choose a level to analyze a given mesh, while allowing to speed-up the method.

3.3 Computation of persistent homology

The last step of our method is the computation of persistent homology of the simplied mesh SM.

We order the cells in SM according to the given filtration and obtain the ordered set of cells $\{\sigma_1, \ldots, \sigma_m\}$ such that if $i < j$ then there exist i', j' such that $i' \leq j'$, $\sigma_i \in SM_{i'}$, $\sigma_j \in SM_{j'}$ and σ_j is not in the boundary of σ_i. We then apply Algorithm 2 to compute persistent homology.

The persistence barcode is stored in a list L with the (birth, death) events as follows: if $\sigma \in M_\ell \setminus M_{\ell-1}$ is born and dies entering $\mu \in M_m \setminus M_{m-1}$, then store (birth, death) in L being birth= $h(v_{i_\ell})$ and death= $h(v_{i_m})$.

Finally, bottleneck distance between different filtrations of the same mesh obtained from different values of δ can be computed to measure the effect of the parameter δ in the persistent homology information obtained.

4 Experiments

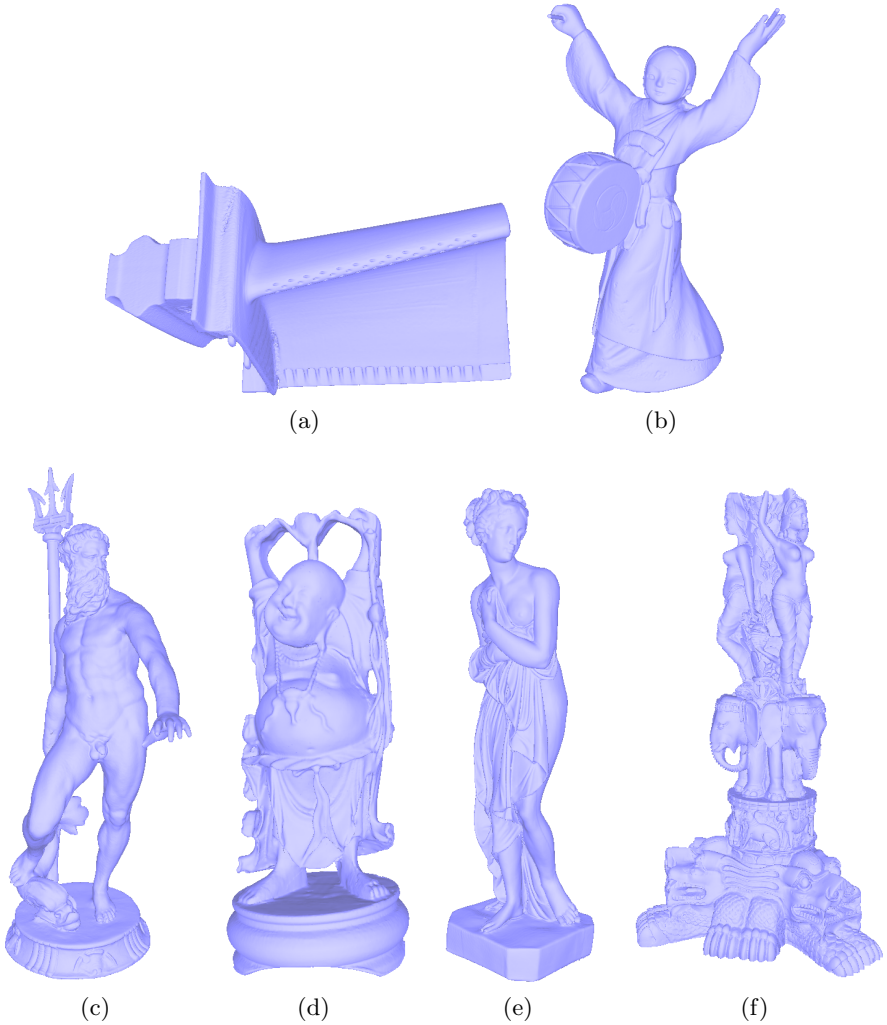
We have implemented our algorithm for persistent homology computation by using the CGAL implementation of combinatorial maps [14] and the additional layer, called linear cell complex, which additionally represents the geometry [15]. All our experiments were run on an Intel®i7-4790 CPU, 4 cores @ 3.60GHz with 32 Go RAM. All the computation times given here are averages of 10 consecutive runs of the same method.

In our tests, we used the six meshes shown in Figure 2, having between $703,512$ and $10,000,000$ faces. All these meshes have only one connected component, except *Blade* which has 295 connected components because it contains many small isolated closed meshes inside the blade.

In our experiment, we compared the persistent homology computation of the six meshes for the following values of δ: 0, 1, 2, 4, 8, 16, 32 and 64. For $\delta = 0$, the persistent homology computed is the one of the lower-star filtration induced by the height function on the vertices of the original mesh. When δ increases, the number of faces in a same cluster increases also and thus the combinatorial map becomes more and more simplified. Nevertheless, persistent homology varies since the filtration varies, although differences are "small".

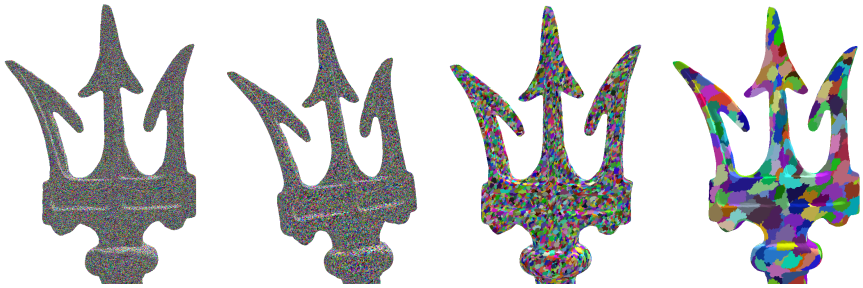
We can see an illustration of the effect of the δ parameter on the size of the different clusters in Figure 3. The number of cells of the different simplified 2-maps for each value of δ is given in Figure 4 (average values for the six meshes).

The effect of δ on the computation time is analyzed in Figure 5 where the six meshes shown in Figure 2 are used, and our method of persistent homology computation based on the 2-map simplification is ran by using different values of δ. Obviously, computation time decreases while δ increases, since more faces belong to the same cluster, and thus the combinatorial map becomes more and more simplified. We can see that the computation time decreases a lot even for small value of δ which is very interesting. For example, for $\delta = 2$, computation time is divided by 2.75 in average.



	#0-cells	#1-cells	#2-cells	#*H0*	#*H1*	#*H2*
(a) Blade	882,954	2,648,082	1,765,388	295	330	295
(b) DrumDancer	1,335,436	4,006,302	2,670,868	1	0	1
(c) Neptune	2,003,932	6,011,808	4,007,872	1	6	1
(d) HappyBuddha	543,652	1,631,574	1,087,716	1	208	1
(e) Iphigenia	351,750	1,055,268	703,512	1	8	1
(f) ThaiStatue	4,999,996	15,000,000	10,000,000	1	6	1

Fig. 2. The six meshes used in our experiments. The table gives the number of i-cells, $\#i$-cells, and the number of Hi generators, $\#Hi$, for $i = 0, 1, 2$.



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

Fig. 3. Effect of the δ parameter on the size of the different clusters for the Neptune mesh, zoom in on the trident. (a) $\delta = 0$. (b) $\delta = 1$. (c) $\delta = 4$. (c) $\delta = 32$.

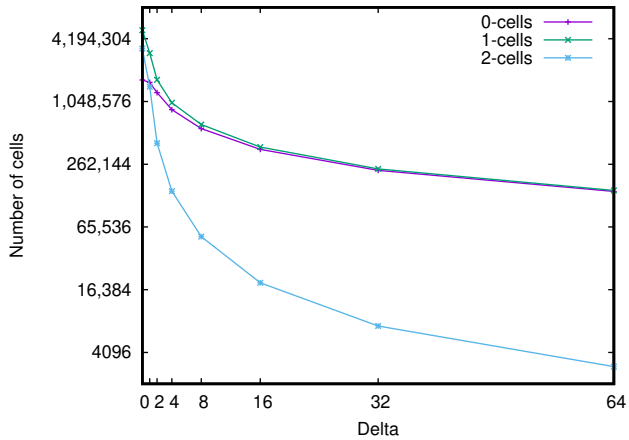
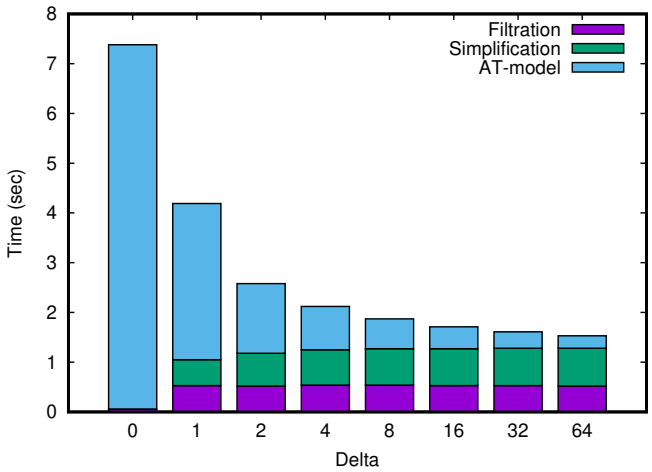


Fig. 4. Number of vertices, edges and faces of the simplified combinatorial maps (in $\log_2$ scale) depending on the value of δ. $\delta = 0$ is the original (non-simplified) 2-map. These numbers are average values for the six meshes.

δ	1	2	4	8	16	32	64
Blade	0.64	1	1.53	2.5	3.43	16.25	10.30
DrumDancer	0.10	0.87	0.62	1.25	1.18	3.31	3.31
Neptune	1.10	1.25	1.67	3.08	5.41	8.00	13.41
HappyBuddha	0.00025	0.0005	0.0014	0.0017	0.0024	0.0060	0.010
Iphigenia	0.88	1.19	1.64	2.71	4.51	9.87	19.22
Statuette	0.90	12.37	12.37	12.24	18.39	26.20	27.41

Table 1. Bottleneck distance between 0-dimensional persistent homology computed on: (1) the lower-star filtration for $\delta = 0$, and (2) the lower-star filtration for different values of δ.



δ	0	1	2	4	8	16	32	64
Blade	13.81	5.94	1.69	1.14	0.93	0.80	0.72	0.69
DrumDancer	2.73	2.37	1.79	1.62	1.48	1.36	1.24	1.19
Neptune	8.42	5.23	3.59	2.92	2.63	2.46	2.29	2.26
HappyBuddha	3.39	1.92	0.99	0.73	0.62	0.56	0.52	0.50
Iphigenia	0.77	0.58	0.40	0.35	0.34	0.31	0.29	0.28
Statuette	13.75	10.13	7.15	6.05	5.30	4.89	4.69	4.49
Average	7.15	4.36	2.60	2.14	1.88	1.73	1.63	1.57

Fig. 5. Computation time (in seconds) of our method by using the patch filtration with increasing δ starting from 0 and going to 64. The graph shows average values for the six meshes, and details time spent in the different parts of the method: computation of the filtration, combinatorial map simplification and persistence computation by using AT-model. The array gives global computation time for each mesh.

We can see in Figure 6 the effect of δ on the results of the persistent homology computation. First, it should be notice that infinite events are always the same whatever the value of δ is. This is a direct consequence of the fact that the homology of the mesh is preserved by our simplification algorithm. For finite events, we can see that their numbers decrease when δ increase. Indeed, the combinatorial map becomes more and more simplified, and thus the number of cells becomes smaller and smaller (as seen in Figure 4).

In Table 1 we can see the bottleneck distance with respect to the 0-dimensional persistent homology between the persitence barcodes corresponding to the lower-star filtration and the filtration obtained when varying δ. Table 2 shows the same information for the 1-dimensional persistent homology. To compute the bottleneck distance we used the package TDA of R[4]. We can observe that, in general, the distance increases when δ increases and the distance is bounded by the value

[4] https://cran.r-project.org/web/packages/TDA/vignettes/article.pdf.

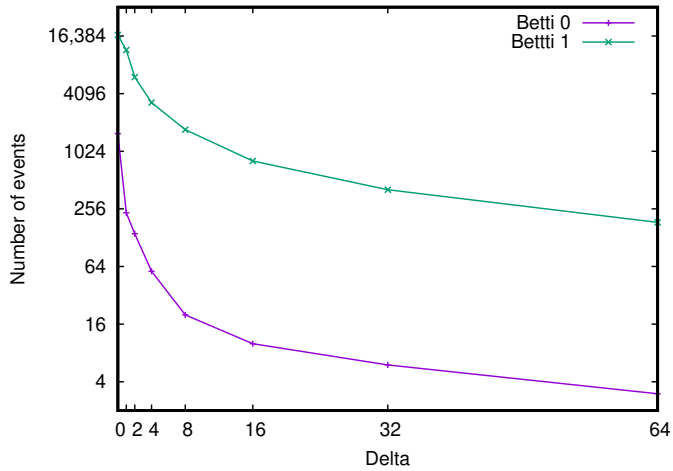


Fig. 6. Number of finite persistence events (in $\log_2$ scale) depending on the value of δ. $\delta = 0$ is the original (non-simplified) 2-map. Betti i is the number of i-homology classes that were born and later died when computing persistent homology, for $i = 0, 1$. These numbers are average values for the six meshes.

of δ. Sometimes, δ increases and the distance is a bit lower. This could occurs due to small pockets in the considered mesh. Moreover we can see that in some meshes the effect of δ is more important than in others. See for example Table 1: for Statuette, the difference between the bottleneck distance for $\delta = 0$ and $\delta = 4$ and for $\delta = 0$ and $\delta = 8$ is only $12.37 - 12.24 = 0.13$ which means that we obtain similar persistent homology information when computing persistent homology using $\delta = 8$ instead of $\delta = 4$. Nevertheless, bottleneck distance for $\delta = 0$ and $\delta = 8$ and for $\delta = 0$ and $\delta = 16$ is $18.39 - 12.24 = 6.14$ which means that we could loss important details if we simplify the mesh using $\delta = 16$ instead that $\delta = 8$.

δ	1	2	4	8	16	32	64
Blade	0.97	1.0	2.0	4.0	7.0	14.0	21.0
DrumDancer	0.14	0.19	0.38	0.58	1.06	2.40	1.89
Neptune	0.53	1.32	1.82	3.15	5.15	8.28	12.41
HappyBuddha	0.00039	0.00067	0.0010	0.0015	0.0028	0.0034	0.005
Iphigenia	0.6	1.2	1.45	2.87	3.59	9.97	7.27

Table 2. Bottleneck distance between 1-dimensional persistent homology computed on: (1) the lower-star filtration for $\delta = 0$, and (2) the lower-star filtration for different values of δ.

5 Conclusion

In this paper, we have defined an algorithm for computing persistent homology of a given filtration defined on a 2D mesh. Persistent homology is computed on different filtrations depending on a parameter δ. When $\delta = 0$, the filtration coincides with the lower start filtration. When $\delta > 0$ the filtration takes, proportionally to δ, more faces in each level. Our method provides high flexibility which allows easily to change the filtration due to the new parameter δ, allowing to speed-up (increasing δ) and giving to users a parameter allowing to tune their results depending on their needs.

One of our future work is to test the different possibilities for clusters regarding to the parameter δ. For example, as one reviewer suggested, it would be interesting not only to take into account the distance between faces but also to consider the height of a face relatively to the seed before adding it to a cluster.

Since we have observed in the experiments that our simplification filters small persistent homology events, we plan to provide theoretical results to this new approach stating that the filtration is stable with respect to δ. That is, the bottleneck distance between two filtrations of the same mesh is bounded by a function on δ. We think we can prove it using the classical result of Edelsbrunner et al on stability of persistence diagrams [4].

Finally, we plan to extend our work to non-orientable manifolds by using the generalized maps package (the non-orientable extension of combinatorial maps) of CGAL. We also would like to define a parallel version of our method: the combinatorial map simplification was already defined in parallel in [7] but we need now to study if it is possible to parallelize some parts of the AT-model computation algorithm. Extension in nD could be given based on the theoretical results for removal and contraction operations in any dimension given in [16, 17]. Indeed, all basic tools used in this work, combinatorial maps, removal / contraction operations and AT-model computation, are defined in any dimension.

Acknowledgments. This research has been partially supported by MINECO, FEDER/UE under grant MTM2015-67072-P. We thank the anonymous reviewers for their valuable comments.

References

1. T. K. Dey, H. Edelsbrunner, S. Guha, Computational topology, in: Advances in Discrete and Computational Geometry, American Mathematical Society, 1999, pp. 109–143.
2. M. W. Bern, D. Eppstein, P. K. Agarwal, N. Amenta, L. P. Chew, T. K. Dey, D. P. Dobkin, H. Edelsbrunner, C. Grimm, L. J. Guibas, J. Harer, J. Hass, A. Hicks, C. K. Johnson, G. Lerman, D. Letscher, P. E. Plassmann, E. Sedgwick, J. Snoeyink, J. Weeks, C. Yap, D. Zorin, Emerging challenges in computational topology, CoRR cs.CG/9909001.
3. G. Carlsson, Topology and data, Bulletin of the American Mathematical Society 46 (2) (1999) 255–308.

4. H. Edelsbrunner, J. Harer, Computational Topology - an Introduction, American Mathematical Society, 2010.
5. D. Günther, J. Reininghaus, H. Wagner, I. Hotz, Efficient computation of 3D Morse-Smale complexes and persistent homology using discrete Morse theory, The Visual Computer 28 (10) (2012) 959–969.
6. V. Robins, P. Wood, A. Sheppard, Theory and algorithms for constructing discrete morse complexes from grayscale digital images, IEEE Trans. Pattern Anal. Mach. Intell. 33 (8) (2011) 1646–1658.
7. G. Damiand, R. Gonzalez-Diaz, Parallel homology computation of meshes, in: Computational Topology in Image Context - 6th International Workshop, CTIC 2016, Marseille, France, June 15-17, 2016, Proceedings, 2016, pp. 53–64.
8. P. Lienhardt, N-Dimensional generalized combinatorial maps and cellular quasi-manifolds, Inte. J. of Computational Geometry and Applications 4 (3) (1994) 275–324.
9. G. Damiand, P. Lienhardt, Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing, A K Peters/CRC Press, 2014.
10. G. Damiand, S. Peltier, L. Fuchs, Computing homology for surfaces with generalized maps: Application to 3d images, in: Advances in Visual Computing, Second International Symposium, ISVC 2006 Lake Tahoe, NV, USA, November 6-8, 2006. Proceedings, Part II, 2006, pp. 235–244.
11. A. Hatcher, Algebraic topology, Cambridge University Press, Cambridge, 2002.
12. R. Gonzalez-Diaz, P. Real, On the cohomology of 3d digital images, Discrete Applied Mathematics 147 (2-3) (2005) 245–263.
13. R. Gonzalez-Diaz, A. Ion, M.-J. Jimenez, R. Poyatos, Incremental-decremental algorithm for computing at-models and persistent homology, in: Computer Analysis of Images and Patterns - 14th International Conference, CAIP 2011, Seville, Spain, August 29-31, 2011, Proceedings, Part I, 2011, pp. 286–293.
14. G. Damiand, Combinatorial maps, in: CGAL User and Reference Manual, 3.9 Edition, 2011, `http://www.cgal.org/Pkg/CombinatorialMaps`.
15. G. Damiand, Linear Cell Complex, in: CGAL User and Reference Manual, 4.0 Edition, 2012, `http://www.cgal.org/Pkg/LinearCellComplex`.
16. G. Damiand, R. Gonzalez-Diaz, S. Peltier, Removal operations in nD generalized maps for efficient homology computation, in: Proc. of 4th Int. Workshop on Computational Topology in Image Context (CTIC), Vol. 7309 of LNCS, Springer Berlin/Heidelberg, Bertinoro, Italy, 2012, pp. 20–29.
17. G. Damiand, R. Gonzalez-Diaz, S. Peltier, Removal and contraction operations in nD generalized maps for efficient homology computation, CoRR abs/1403.3683.