1 Introduction

The Multi-Tesellation for Forman Gradient (MTFG) is a multiresolution model encoding varius representations for a scalar field, defined on a simplicial 2-complex, both from the geometrical and topological point of view. The Multiresolution model is encoded as a Direct Acyclic Graph (DAG).
The root of the DAG contains:

- the base complex encoded through the IA data structure [6]
- a Forman Gradient defined on such complex encoded through the compact representetion described in [8] which requires only 2 bytes per top simplex.

Nodes of the DAG represents updates which are geometrical or topological. Geometrical updates are the undo of an edge-contraction [5], that we will call edge-expansion. An edge-expansion introduces at most two triangles, tree edges and a new vertex in the simplicial complex. A topological refinement defines two simplexes as critical and can be considered as an undo of a removal operator [1].

2 MTFG, required items

In this Section we will describe the steps required to build up a MTFG starting from a simplicial complex given in input. As first step, the simplicial complex Σ is encoded using the IA data structure which ecplicitly represents only the vertices V and the triangles T. For each vertex its coordinates are stored as well as one of its incident triangles. For each triangle instead, we store the three vertices on its boundary and the adjacent triangles (working with manifold meshes they will be at most one for each boundary edge).

Then, the Forman Gradient F is computed on Σ using the algorithm described in [7]. Following the encoding described in [8] F will be stored as a set of gradient configurations, one for each triangle in T. Once the simplicial complex and the corresponding Forman gradient are properly stored they are simplified as much as possible in order to obtain a coarser simplicial complex and its corresponding Forman gradient which will become the base complex Σ_B stored in the DAG root.

2.1 Edge-collapse

And edge-collapse $collapse(v_1, v_2)$ is a well known simplification operation for simplicial complexes which deletes an edge collapsing one of its vertices in the other one, first vertex is deleted as well. According to the notation showed in Figure 2.1(a) we will indicate with:

- v_1 the vertex that will be deleted by the edge-collapse,

- v_2 the vertex that will survive to the edge-collapse,
- $e = (v_1, v_2)$ the edge to collapse,
- t^{left} the triangle on the left of e to delete,
- t^{right} the triangle on the right of e to delete,
- v_3^{left} and v_3^{right} the vertices in t^{left} and t^{right}, respectively, different from v_1 and v_2,
- t_1^{left} and t_1^{right} the triangles adjacent to t^{left} and t^{right}, respectively, on the edge opposite to v_2,
- t_2^{left} and t_2^{right} the triangles adjacent to t^{left} and t^{right}, respectively, on the edge opposite to v_1.

Even if an edge collapse can be performed on any edge of Σ we want to mantain two invariants across our simplification process. Each simplification has to be *homology-preserving* and must not produce *invalid gradient configurations.*

Homology-preserving edge collapse In general an edge-collapse is not guaranteed to preserve the homology of the simplicial complex on which it is applied. This is a well studied problem in literature [5] and in our implementation we use a variant of the so called *Link Condition* described in [5] for a 2-manifold simplicial complex. We extract all the vertices adjacent to v_1 and v_2, if the size of this two sets is equal or lower than two, the link condition is mantained and the edge collapse is homology-preserving.

Valid Gradient configuration Second fundamental invariant required is the validity of F. Removing simplexes from Σ we have to modify F accordingly in the neighborhood of the edge collapsed. The key idea of a gradient update is to locally modify F not generating new critical simplexes and mantaining the gradient flow. To guarantee this, we have to specify some preconditions for the gradient in the neighborhood of the edge deleted. The edge collapse is applicable if:

- t^{left} and t^{right} are not critical,
- all the edges on t^{left} and t^{right} are not critical,
- v_1 has more than three incident triangles,
- v_1 is paired with the edge e in F,
- v_3^{left} is not paired in F with the edge (v_3^{left}, v_1),
- v_3^{right} is not paired in F with the edge (v_3^{right}, v_1).

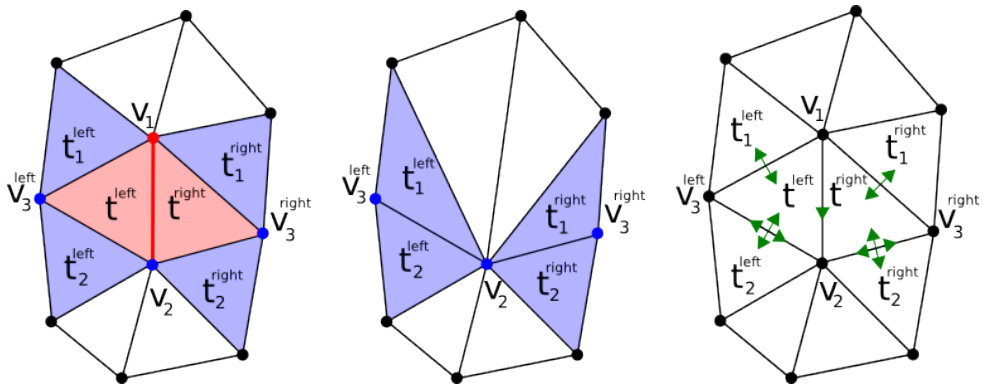

Figure 1: In (a) are shown the simplexes involved in a edge-collapse. In (b) the result of the edge collapse and in (c) are shown the gradient arrows (green) that are lecit for the edge-collapse.

In Figure 2.1(c) we show all the possible gradient arrows that can appear in the neighborhood of an edge-collapse. Gradient arrows not showed in the figure are pairing simplexes not involved in the edge collapse or are invalid arrows for the collapse operation.

The edge-collapse applied on the edge e simplify the simplicial complex locally. As showed in Figure 2.1(b) it deletes the edge e collapsing vertex v_1 in v_2. Triangles t^{left} and t^{right} are deleted as well. As a consequence,

- t_1^{left} become adjacent to t_2^{left} (update TT relation)
- t_1^{right} become adjacent to t_2^{right} (update TT relation)
- all the triangles incident in v_1 switch vertex v_1 with v_2 (update TV relation)
- if v_3^{left} or v_2 were storing t^{left} as incident triangle, it is switched with t_1^{left} or t_2^{left} (update VT* relation)
- if v_3^{right} or v_2 were storing t^{right} as incident triangle, it is changed with t_1^{right} or t_2^{right} (update VT* relation).

Thanks to the precoditions, the Forman gradient has to be updated only on a limited number of simplexes. We have to update the gradient codified in t_1^{left} and t_1^{right} in the worst case. Since the updates required are simmetrical on the left and on the right part, respect to the collapsed edge, we will describe them for the left side only.

The updates on F depend on the edges $(v_3^{left}, v2)$ and $(v_3^{left}, v1)$. If these edge are both paired with a triangle then the edge-collapse require no updates

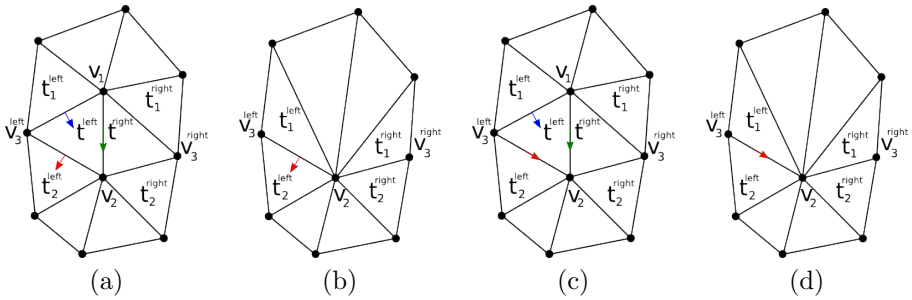

Figure 2: In (a) and (b) is shown a gradient configuration that does not require an update of F during an edge collapse. Triangle t_1^{left} has an arrow pointing the adjacent triangle before (blue arrow in (a)) and after (red arrow in (b)) the edge collapse. On the countrary, if the gradient configuration is different (in (c)) it losts the arrow pointing the adjacent triangle after the edge collapse (see (d))

on F, (see Figure 2.1(a) and 2.1(b)). Otherwise, if the edge $(v_3^{left}, v1)$ is paired with a vertex (v_1 or v_3^{left} does not matter) this gradient pair must be updated in the graient codified in t_1^{left}, (see Figure 2.1(c) and 2.1(d)).

In the implementation, are performed all the updates on the gradient first, then the two triangles t^{left} and t^{right} and the vertex v_1 are removed and the IA data structure is modified updating the new adjacencies between triangles.

2.2 Edge-collapse by batch

Edge-collapse simplifications are the first tool we use to reduce the resolution of Σ and, consequently, its storage cost. Our goal is to apply as mush edge-collapses as possible and, at the same time, to distrubute them throughout the dataset in order to decrease the resolution uniformly. To achieve this result, we use a simplification algorithm that we will call by batch. Given Σ we enqueue all the performable edge-collapses in a priority queue in ascending order of edge lenght. The first collapse is performed and all the vertices adjacent to the removed one are marked as visited. Then, a collapse is performed only if the two vertices on the edge are not yet visited. Once the priority queue is empty a new one is built with all the new performable edge-collapse and all the vertices are marked has unvisited. The edge-collapse by batch ends when no more collapses are performable.

The definition of a simplification algorithm is a fundamental step for the construction of a multiresolution-model. As discussed in other works [3], different simplification algorithms infer different properties on the DAG that will codify the multiresolution model. For the MTFG we have chosen a batch simplification maximizing the number of independent simplifiations (and thus of

independent refinements) in order to maximize the number of representations that can be extracted.

2.3 Topological simplification

The second type of simplifications we want to perform on the scalar field are topological. We will use two types of simplification operators defined in [1] to reduce the number of critical points in the scalar field, these operators are called $removal_{i,i+1}$ and $removal_{i,i-1}$. A removal operator remove two critical simplexes from a Forman gradient changing locally the flow of F. We will not discuss here the updates that this operator causes on the scalar field but we will describe (briefly) only the updates in terms of Forman gradient.

Given a Forman gradient F and a $removal_{i,i+1}(q, p, p')$, where q is a critical i-simplex connected with the two critical $(i+1)$-simplexes p and p' by two single $(i+1)$-paths on F, we call F' the Forman gradient obtained applying $removal_{i,i+1}(q, p, p')$ on F.

To obtain F' critical simplexes q and p are removed by the set of critical simplexes and the (i+1)-path connecting p to q is reversed. Formally, given a path

$$p = \tau; (\sigma_1, \tau_1); (\sigma_2, \tau_2); ...; (\sigma_n, \tau_n); \sigma = q$$

where (σ_i, τ_i) is a pair in F and σ_i is a simplex on the boundary of τ_{i-1}, we will obtain the following gradient path

$$(\sigma_1, \tau); (\sigma_2, \tau_1); ...; (\sigma_n, \tau_{n-1}); (\sigma, \tau_n)$$

.

This kind of simplifications cause an update of the gradient encoded only while no updates are performed on the simplicial complex (IA data structure). An example of the updates performed on the gradient are shown in Figure 2.3 for a 2-path (a) and (b) and for a 1-path (c) and (d).

2.4 Topological simplification by batch

Similarly to the geometrical simplifications, we want to perform as much topological simplifications as possible distributing them throughout the dataset. To achieve this result we use a support data structure encoding the incidence graph (IG) of the critical points of the scalar field. We will not described the IG in details. A complete description can be found in [3] where is illustrated how the IG provides a compact representation of the scalar field as well as a representation for the incidence relations between the cells of the Morse complexes.

The IG is extracted from F and is kept up to date during the undergoing of topological simplifications. The simplification algorithm that we use is the same used in [2]. At the beginning a priority queue is built with all the valid topological simplifications performable. When a simplification is performed all

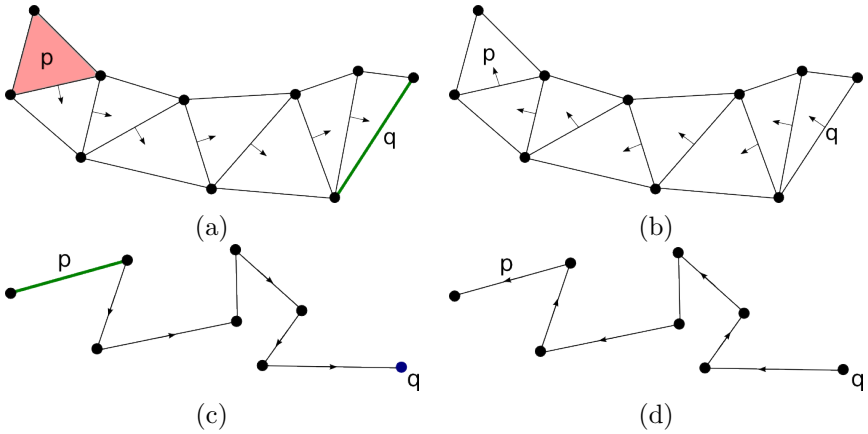

Figure 3: In (a) an example of a 2-path connecting a critical triangle with a critical edge. In (b) the 2-path is inverted deleting the two critical simplexes. In a similar fashio in (c) a 1-path is shown and in (d) the same 1-path is inverted deleting the critical vertex and the critical edge.

the nodes involved are marked as visited and no other simplification will be performed on them until other simplifications are available. When the priority queue is empty it is refilled with new simplifiations unmarking all the nodes in the IG, and when no more simplifications are available the algorithm ends.

2.5 Building the root of the MTFG

Building an $MTFG$ from a simplicial complex Σ is a procedure that alternates geometrical and topological simplification algorithms. As first step the incidence graph IG is computed on F. Then all the performable edge-collapses are executed until all the edges present an invalid configuration to be collapsed. Thus the queue of the topological simplifications is prepared and a percentage of all the possible topological simplifications is performed. This percentage can be a user-defined parameter. We have fixed this parameter based on the persistence range among all the possible simplifications. At each iteration are performed all the simplifications having persistence value lower or equal than the 10% of the maximum persistence value. Performing some topological simplifications will unlock new edge-collapses (since some critical simplexes are removed and some arrows are flipped). This interchange of geometrical and topological simplifications continues until no more simplifications are available.

All the information regarding the simplifications executed are stored in two lists, one for the geometrical and one for the topological. These will be the chunk used to build up the DAG structure of the MTFG. The root of our DAG is built, at this point, storing the base simplicial complex Σ_B obtained performing all the edge-collapse and the corresponding base Forman gradient F_B.

3 Building the MTFG and its storage cost

Once all the simplifications have been performed and stored build up a MTFG corresponds to create the DAG nodes (one for each simplification) connecting them accordingly to a dependency relation. In this Section we will describe the two types of DAG nodes, *geometrical DAG node* ($node_{geom}$) and *geoemtrical DAG node* ($node_{topo}$), and we will describe the dependency relations among them.

3.1 Geometrical DAG Node

A geometrical DAG node encodes the edge expansion refinement, undo of an edge-collapse $collapse(v_2, v_1)$. A geometrical DAG node $node_{geom}$ representing an edge expansion encapsulates:

- a vector vv of vertices adjacent to v_2 when the $collapse(v_2, v_1)$ was applied,
- the coordinates of the new vertex inserted v_2
- an integer v indicanting the vertex v_1 that will be on the boudnary of the new edge with v_2,
- a set of $node_{geom}$ from which this operation depends (see Section 3.3).
- a boolean value indicating wether the expansion has been applied or not.

Vertices in vv are stored in a precise orded such that in the firsts two positions of the vector will be stored the two vertices v_3^{left} and v_3^{right} (see Figure 2.1(a)). Memory occupied by vertex v_2 instead is used differently before and after the expansion is executed. If the expansion is not yet applied integer v indicates the index of vertex v_1. After the expansion, its value will be replaced with the index of the new vertex v_1. This is an implementation trick to have an access (with complexity O(1)) to a vertex during the extraction of the front complex Σ'

Implementation explanation All the indices of a vertex in the MTFG are store as signed integers. We have to distinghuish between vertices codified in the base complex Σ_B and vertices that will be introduced by some $node_{geom}$. To do this, we store all the $node_{geom}$ in consecutive memory cells. Then, we distinguish among this two kind of vertices representing them as:

- a negative integer $-i$, if the index correspond to the i-th vertex in Σ_B,
- a positive integer i, if the index corresponds to the vertex introduced by the i-th $node_{geom}$. If the expansion codified in $node_{geom}$ has been applied, v represent the actual index of the vertex in the front complex Σ'.

The refinement will reintroduce a vertex v_2 as well as a new edge e and the two incident triangles t^{left} and t^{right}. (See Figure 3.1(b)). The updates required on the IA data structure are:

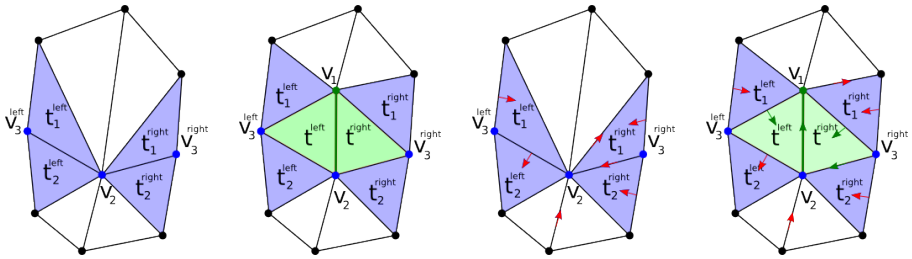

Figure 4: In (a) is shown a set of simplexes before the edge expansion showed in (b). In (c) and (d) the same edge expansion is illustrated showing the updates required on the Forman gradient with two different configurations on the left and on the right of the introduced edge.

- add a new vertex v_1 in the set of vertices,
- add a new triangle t^{left} composed by the three vertices (v_1, v_2, v_3^{left})
- add a new triangle t^{right} composed by the three vertices (v_1, v_2, v_3^{right})
- set the adjacenct triangles of t^{left} as $(t_1^{left}, t_2^{left}, t^{right})$
- set the adjacenct triangles of t^{right} as $(t_1^{right}, t_2^{right}, t^{left})$
- all the triangles between t^{left} and t^{right} (visited in cw order) get v_1 as incident vertex instead of v_2.

Also the front Forman gradient F' has to be modified in the neighborhood of the introduced simplexes. Similarly to the collapse, we can unambiguosly determine how to extend the gradient on the new simplexes simply knowing the gradient defined on the actual simplexes.

Updates required on the front forman gradient F' are the following (as before we will describe the updates only for the left side):

- new edge edge e is paired in F' with v_1 or v_2. If v_2 is paired with an edge that will be redirected to v_1, than e is paired with v_2 otherwise it is paired with v_1. (See Figure 3.1(c) and (d))
- if edge (v_3^{left}, v_2) is paired with v_2 or v_3^{left}, then edge (v_3^{left}, v_1) is paired with t^{left}
- if edge (v_3^{left}, v_2) is paired with t_2^{left}, then edge (v_3^{left}, v_1) is paired with t^{left}
- if edge (v_3^{left}, v_2) is paired with t_1^{left}, then edge (v_3^{left}, v_2) will be paired with t^{left} and edge (v_3^{left}, v_1) will be paired with t_1^{left}.

3.2 Topological DAG Node

A topological DAG node encodes the refinement applicable on an intermediate Forman gradient F', undo of a removal. A topological DAG node $node_{topo}$ representing a refinement encapsulates:

- the set of vertex indices indicating the simplex that will be critical (a triangle or a point)
- the pair of vertex indices indicating the edge that will be critical
- a set of $node_{topo}$ from which this operation depends (see Section 3.3).
- a set of $node_{geom}$ from which this opeartion depends (see Section 3.3)

As undo of a removal the refinement introduce two critical simplexes reversing the gradient arrows in order to create a path from the simplex of higher dimension to the simplex of lower dimension. Is not necessary to codify the path complitely since, knowing the correct starting simplex the path to reach the ending critical simplex is unique. There is no need to store any other information on the other critical simplexes in the neighborhood, required from the operation, but they are implicitly encoded in the dependency relations

To follow a unique path we always need to start from the critical simplex of higher dimension. We will describe here the two different paths followed in case the operation is an $insert_{i,i+1}$ or an $insert_{i,i-1}$.

$insert_{i,i+1}$ In the case of 2-complexes this will correspond to the introduction of a critical edge e and a critical triangle t, saddle and maximumum critical points respectively. The triangle t is identified and the edge e_1 with which is paired is stored. Then t is set as critical and the e_1 is used to navigate to the adjacent triangle of t, t_1. At this point, having e_1 and t_1 we proceed recursively:

- e_t is the pair of t_1 in F'
- we make a new pair in F' (e_1, t_1)
- $e_1 = e_t$
- $t_1 =$ adjacent of t_1 on edge e_t

This corresponds to swap the arrows until e_t is equal to e. Then, e_t is set as critical and the algorithm ends.

$insert_{i,i-1}$ In the case of 2-complexes this will correspond to the introduction of a critical edge e and a critical vertex v, saddle and minimum critical points respectively. The critical vertex v is identified and the edge e_1 with which is paired is stored. Then v is set as critical and e_1 is used to navigate to the adjacent vertex v_1. At this point, having e_1 and v_1 we proceed recursively:

- e_t is the pair of v_1 in F'
- we make a new pair in F' (v_1, e_1)
- $e_1 = e_t$
- v_1 = vertex adjacent of v_1 on edge e_t

This corresponds to swap the arrows until e_t is equal to e. Then, e_t is set as critical and the algorithm ends.

3.3 Dependency relations

The dependency relation between two DAG nodes express a constrained order in which two opertions must be performed. We can indentify three types of dependency relations: dependecy between geometrical DAG nodes, dependency between topological DAG nodes and dependency of a topological DAG node from a geometrical DAG node.

Geometrical vs Geometrical Intuitively, a geometrical DAG node n_1 depends from a geometrical DAG node n_2 when n_2 introduce a vertex used by n_1. In our work we set the dependency based on the set of vertices vv (see Section 3). So formally, n_1 depends from n_2 if n_2 introduce a vertex present in the set vv of n_1.

Topological vs Topological Intuitively, a topological DAG node n_1 depends from a topological DAG node n_2 when n_2 introduce a critical simplex that is involved in the neighbrohood of n_1. This can be easily noticed considering the operations on the IG instead of on the Forman gradient F (see [4]). Thus these information are stored during the simplification process, saving, for each node deleted from the IG which DAG represent its deletion. Once the hierarchy is built these information are used to connect the different topological DAG nodes with arcs in the DAG and information on the nodes of the IG are no longer required.

Topoligical vs Geometrical Each topological DAG node can be also dependent from a geomtrical DAG node. In particular to execute a topological refinement the two critical simplexes have to exist in the simplicial complex and in particular all the vertices composing them, must be present in Σ'. We can assure that, if all the vertexes composing the required simplexes, are inside the simplicial complex, then the two critical simplexes are inside the simplicial complex as well.

From these set of dependencies we can see that a geometrical refinement is always independent from the topological resolution while a topological refinement is also linked to a specific resolution level for the geometry. This is the

reason why, during the simplification step, we always perform as much geometrical simplification as possible while we perform only a subset of the topological operation performable at each operation. This way we can fix the lowst level of resolution required for each topological operation.

3.4 Storage cost

We can estimate the storage cost for the MTFG considering the storage cost of the IA data structure for the simplicial base complex Σ_B, the storage cost of the base Forman gradient F_B, the storage cost for the single DAG nodes and the arcs. We are estimating the storage cost of a pointer as $4byte$.

A geometrical DAG node occupies $4byte$ for each vertex in vv, $32byte$ for the coordinates of v_2 (4 floats), $4byte$ for vertex v_1 and $4byte$ for each $node_{geom}$ pointer and 1 byte for the boolean. Thus $37 + 4|vv| + 4|node_{geom}|$ byte.

A topological DAG node occupies $4byte$ for each vertex used to indicate the critical simplexes (depending from the opertion we need 6 or 4 vertices), $4byte$ for each $node_{topo}$ and $4byte$ for each $node_{geom}$. Thus overestimating the amount of vertices needed, $24 + 4|node_{topo}| + 4|node_{geom}|$, we can overestimate the number of $node_{geom}$ (one for each vertex) and thus it will cost in total $48 + 4|node_{topo}|$ bytes.

The Σ_B as described in [6] occupyes $35|V|+24|T|$ where $|V|$ and $|T|$ represent the number of vertices and triangles respectively.

4 Selective refinement on a MTFG

Once we have built the $MTFG$ hierarchical structure we can extract representations at different level of details both from the topological and geometrical point of view. Firstly we have to set the desired resolution from the topological point of view, thus the first input value will be the minimum persistence value desired inside the complex. The topological refinements performed will include also some geometrical refinement since, some of the topological operations will be dependent by some edge-expansion.

Once the desired persistence level is reached, the geometry can be refined as well. In a uniform selective refinement query the minimum persistence and the minimum edge-lenght are set throughout the dataset uniformely.

However, more interesting queries can be performed thanks to the DAG structure of the multiresolution model, we call these queries *extraction at variable resolutions*. In this kind of queries different resolutions levels are set, for both the persistence and the edge-lenght values, across the dataset. For the topological resolution then, two values are given in input, first one indicates the desired resolution inside the region of intereset (a box or a sphere inside the domain) while the second value indicates the desired resolution outside the region of interest. Once the topological resolution has been increased at wish,

two new values are given in input to increase the geometrical resolution in a similar fashion. The region of interest used for the geometrical resolution can be the same used for the topological refinements or a new one. It can also be dependent from topological properties and not be set in a particular region. Given a persistence value p, for example, we can increase to resolution r all the simplexes belonging to a Morse cell having persistence value greater or equal than p [not yet implemented].

References

[1] Lidija Comic and Leila De Floriani. Dimension-independent simplification and refinement of morse complexes. *Graphical Models*, 73(5):261–285, 2011.

[2] Lidija Comic, Leila De Floriani, and Federico Iuricich. Simplifying morphological representations of 2d and 3d scalar fields. In Isabel F. Cruz, Divyakant Agrawal, Christian S. Jensen, Eyal Ofek, and Egemen Tanin, editors, *GIS*, pages 437–440. ACM, 2011.

[3] Lidija Comic, Leila De Floriani, and Federico Iuricich. Multi-resolution cell complexes based on homology-preserving euler operators. In Rocío González-Díaz, María José Jiménez, and Belén Medrano, editors, *DGCI*, volume 7749 of *Lecture Notes in Computer Science*, pages 323–334. Springer, 2013.

[4] Lidija Comic, Leila De Floriani, and Federico Iuricich. Simplification operators on a dimension-independent graph-based representation of morse complexes. In Cris L. Luengo Hendriks, Gunilla Borgefors, and Robin Strand, editors, *ISMM*, volume 7883 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2013.

[5] Tamal K. Dey, Anil N. Hirani, Bala Krishnamoorthy, and Gavin Smith. Edge contractions and simplicial homology. *CoRR*, abs/1304.0664, 2013.

[6] G. M. Nielson. Tools for triangulations and tetrahedralizations and constructing functions defined over them. In G. M. Nielson, H. Hagen, and H. Müller, editors, *Scientific Visualization: overviews, Methodologies and Techniques*, chapter 20, pages 429–525. IEEE Computer Society, Silver Spring, MD, 1997.

[7] Vanessa Robins, Peter John Wood, and Adrian P. Sheppard. Theory and algorithms for constructing discrete morse complexes from grayscale digital images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(8):1646–1658, 2011.

[8] Kenneth Weiss, Federico Iuricich, Riccardo Fellegara, and Leila De Floriani. A primal/dual representation for discrete morse complexes on tetrahedral meshes. *Comput. Graph. Forum*, 32(3):361–370, 2013.