Computers & Graphics (2018)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

A Heat Flow Based Relaxation Scheme for n Dimensional Discrete Hyper Surfaces

Marco Livesu[a,*]

[a]*CNR IMATI, Genoa, Italy*

ARTICLE INFO

Article history:
Received February 7, 2018

Keywords: Diffusion, Smoothing, Implicit hyper surfaces, Heat flow

ABSTRACT

We consider the problem of relaxing a discrete $(n-1)$ dimensional hyper surface defining the boundary between two adjacent n dimensional regions in a discrete segmentation. This problem often occurs in computer graphics and vision, where objects are represented by discrete entities such as pixel/voxel grids or polygonal/polyhedral meshes, and the resulting boundaries often expose a typical jagged behavior. We propose a relaxation scheme that replaces the original boundary with a smoother version of it, defined as the level set of a continuous function. The problem has already been considered in recent years, but current methods are specifically designed to relax curves on triangulated discrete 2-manifolds embedded in $\mathbb{R}^3$, and do not clearly scale to multiple discrete representations or to higher dimensions. Our biggest contribution is a smoothing operator entirely based on three canonical differential operators: namely the Laplacian, gradient and divergence. These operators are ubiquitous in applied mathematics, are available for a variety of discretization choices, and exist in any dimension. To the best of the author's knowledge, this is the first intrinsically dimension-independent method, and can be used to relax curves on 2-manifolds, surfaces in $\mathbb{R}^3$, or even hyper-surfaces in $\mathbb{R}^n$. We demonstrate our method on a variety of discrete entities, spanning from triangular, quadrilateral and polygonal surfaces, to solid tetrahedral meshes.

1. Introduction

Labeling (or *segmenting*) an object is a fundamental operation in computer graphics and vision, widely used in applications such as analysis of medical images, object recognition and detection.

The majority of segmentation algorithms work on discrete domains, such as regular grids [1], polygonal [2, 3, 4, 5] and polyhedral [6, 7] meshes. A common approach consists in assigning to each element of the domain a value (or *label*). Elements sharing the same label belong to the same region, whereas elements with different labels belong to separate regions. As a consequence, boundaries between adjacent regions are only intrinsically defined, and amount to the union of the interfaces between adjacent elements.

Given a n dimensional object and a labeling defined on it, the boundary between a pair of adjacent regions is a $(n-1)$ dimensional hyper surface. As a concrete example one may consider a binary partition of a discrete two dimensional surface (e.g. a triangle mesh): the boundary between the two regions is the chain of edges having polygons with opposite labels at its sides. The same goes for three-dimensional objects (e.g. a tetrahedral mesh): the boundary is the set of faces having polyhedra with opposite labels at its two sides. Indeed, the boundary is one dimensional if the object is two dimensional, and is two dimensional if the object is three dimensional.

Depending on the quality of the discretization, both in terms of number, regularity, and shape of each element in the domain, the boundaries between adjacent regions can be geometrically poor, showing a typical jagged behaviour (Figure 1). In this

*Corresponding author: Tel.: +39-010-64-75-624;
e-mail: marco.livesu@gmail.com (Marco Livesu)

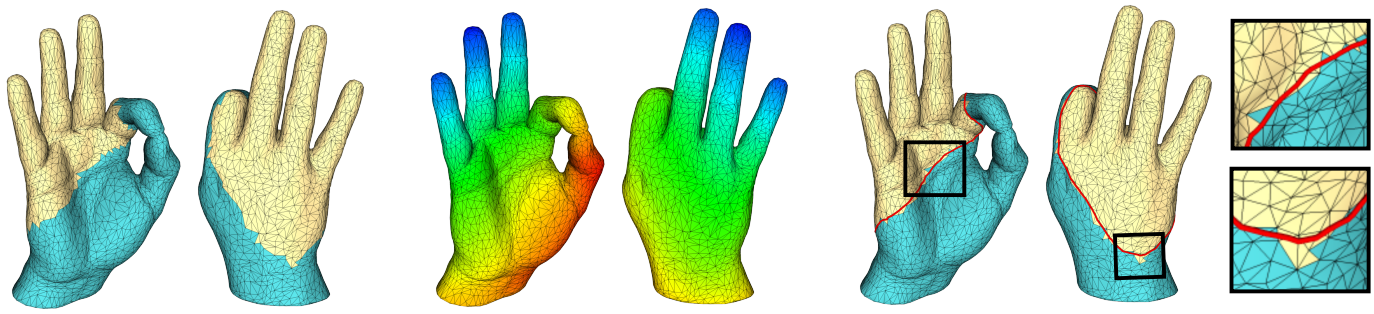

Fig. 1. Boundaries between adjacent regions in discrete segmentations can be geometrically poor, exposing a typical jagged behaviour (left). We relax discrete boundaries by approximating them with level sets (right, red curves) of continuous functions (middle).

article we focus on this very specific problem, and propose a neat method to relax jagged discrete boundaries, replacing them with smooth, yet geometrically faithful, versions of them.

Our most important contribution is a method which is completely agnostic both on the discrete representation used for the domain, and on the dimension of the space in which it operates. To do so, we define our smooth hyper surfaces as level sets of continuous functions. We take inspiration from [8], and generate such functions relying on three classical discrete differential operators: namely the Laplacian, the gradient and the divergence. These operators are ubiquitous in applied mathematics, have been implemented for a variety of discretization choices, and exist in any dimension.

Our method is efficient and easy to implement. We believe it has great potential not only for classical applications, like refining the boundaries of a discrete segmentation, but also for applications like data mining, where clustering problems in high dimensional spaces often occur, and the refinement of the clusters' boundaries may be beneficial for classification algorithms.

This article extends [9], providing: (i) a more general formulation to compute gradients of polygonal and polyhedral meshes;(ii) a discussion on some connectivity issues that may arise in discrete meshes and produce unpredictable behaviour; (iii) a wider set of examples on meshes of various type and dimensionality; (iv) a study on performances (running times); (v) a study on how user parameters influence the smoothing operator.

2. Related Works

The problem of having jagged boundaries in discrete segmentations is well known in literature. Most of the segmentation algorithms are not capable of producing smooth boundaries [10]. An exception are the concavity-aware segmentations [11], which exploit harmonic functions to natively generate smooth cuts. Some other methods achieve boundary smoothness in post processing, implementing additional steps in their pipeline [12, 13]. Whenever the segmentation algorithm of choice is not capable of producing smooth boundaries by itself, smoothness between adjacent regions can be achieved using third party algorithms. The method discussed in [14] is very similar in spirit to our approach, as it is based on the level sets of continuous functions. However, it produces smooth curves that tend to *escape* from their original position (see Figures 1a and 4b from the original paper). Our method produces smooth boundaries that faithfully follow their discrete counterparts, improving geometric fidelity. Panozzo *et al.* [15] introduced a method to project B-Spline curves on discrete surfaces using the Phong projection. This tool could in principle be used to define smooth segmentation curves, but the user should define and place the control points that define the smooth curve. Iawonn and colleagues [16] proposed an iterative Laplacian smoothing algorithm that at each iteration reduces the local boundary curvature. In [17] a local parameterization method for defining geometric features in triangle meshes is proposed. The method is based on the concept of *snakes*, pioneered by Kass and colleagues in [18]. All these methods specifically address discrete 2-manifolds embedded in $\mathbb{R}^3$, and do not directly extend to higher dimensions. To the best of our knowledge ours is the first intrinsically dimension-independent method, and can be used to relax discrete curves on 2-manifolds, surfaces in $\mathbb{R}^3$, or even hyper-surfaces in $\mathbb{R}^n$.

Diffusion. Solutions to the heat equation have been extensively used in computer graphics and vision to compute segmentations [19], geodesic distances [8, 20], Voronoi diagrams [21] and compare shapes [22]. An in depth review of the applications and computational methods used to compute diffusion distances is beyond the scope of this paper. We point the reader to [23] for a recent survey on this topic.

3. Method

Our method inputs a discrete object and a segmentation that partitions it in two separate components. The output is a smoothed version of the discrete boundary that separates the two components, defined as the level set of a continuous function. The goal is therefore the generation of a smooth function that roughly aligns with the discrete segmentation boundary. To do so, we first solve a diffusion problem that propagates some heat from the boundary to the rest of the domain. We then start from the gradient of the heat flow and we process it in order to design the gradient of our target function. We eventually integrate the resulting vector field to produce such function.

A careful reader may notice that this work is heavily inspired from [8], where a similar approach was used to compute geodesic distances on discrete meshes. Indeed, the basic steps of the algorithm are the same, although for geodesic distances

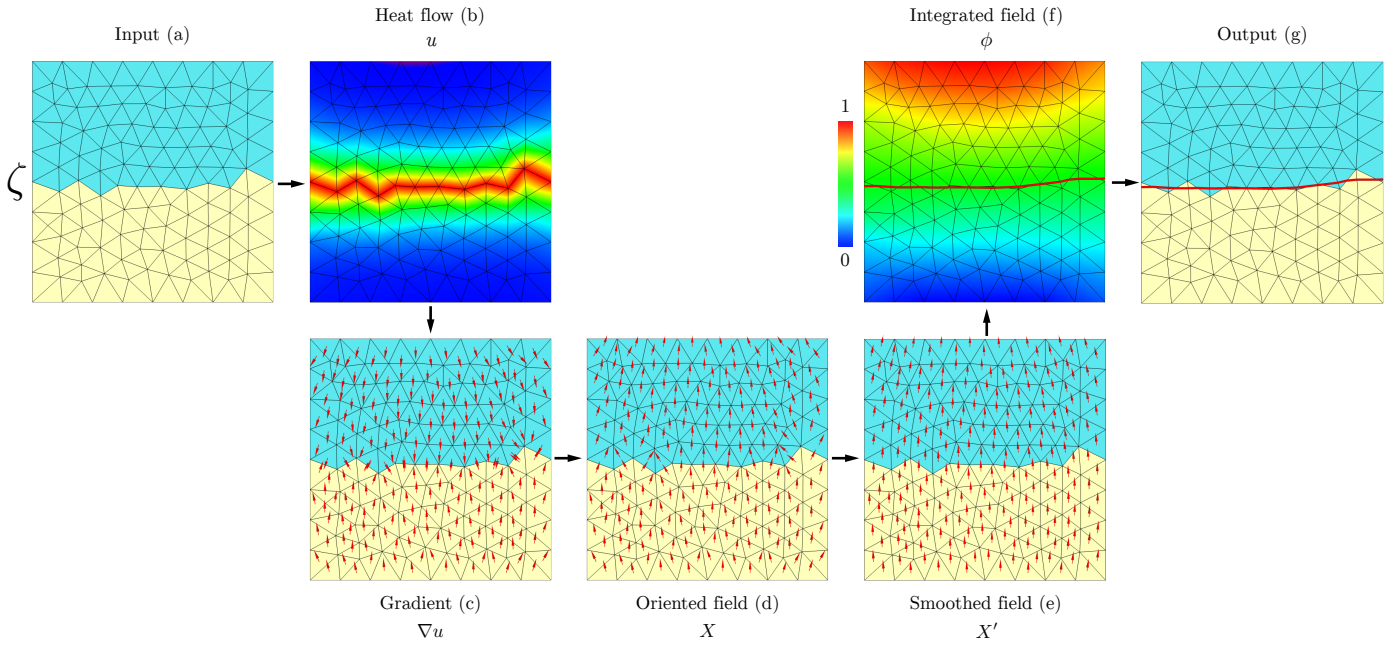

Fig. 2. Overview of our method: (a) we input a discrete segmentation with a jagged boundary ζ; (b) heat diffuses from ζ, producing a function u; (c,d) the vector field X is defined as $\nabla u/\|\nabla u\|$ on the yellow region, and $-\nabla u/\|\nabla u\|$ on the blue region; (e) the field X is smoothed, producing a new vector field X'; (f) the field X' is integrated, producing a function ϕ with level sets that globally align to ζ; (g) the output smoothed boundary is defined as a level set of ϕ.

the target function is the Eikonal equation, whereas in this article we target a completely different function, and so the way we process the gradient of the heat flow and integrate it to produce our target function is completely different.

For the sake of clarity and ease of illustration we introduce the algorithm considering as input a surface mesh representing our discrete domain, and a chain of edges representing a discrete curve ζ defined on it (Figure 2). We also assume that ζ partitions the domain in two disjoint regions. We remind the reader that as long as the Laplacian, gradient and divergence operators are available, the same algorithmic steps apply to any dimension or discretization choice.

The hyper-surface smoothing method can be decomposed in the following four distinct algorithmic steps:

1. Diffuse the heat $\dot{u} = \Delta u$ starting from ζ for some time t (Figure 2b);

2. Initialize the vector field X as $\nabla u/\|\nabla u\|$ in one region, and $-\nabla u/\|\nabla u\|$ in the other region (Figure 2d);

3. Smooth X, producing a new vector field X' (Figure 2e);

4. Integrate X' in the least squares sense, producing a function ϕ that aligns to X' and has level sets that globally align to ζ (Figure 2f).

The output of the algorithm is the function ϕ, and the smoothed contour will then be a level set of it (Figure 2g).

In the remainder of the section we provide more information on each algorithmic step, also discussing implementation details regarding the discretization of differential operators. In particular, we propose a convenient unified representation for the gradient and divergence operators which applies to any polygonal and polyhedral mesh.

Step 1: heat flow. The first step of the algorithm consists in placing heat charges along the boundary ζ, letting them diffuse over time, according to the flow

$$\frac{\partial u}{\partial t} = \Delta u. \tag{1}$$

We perform implicit time integration using the backward Euler method, as described in [24]. This amounts to solving the following equation

$$(I - t\Delta)u = 0_{\big|_{\zeta=1}}. \tag{2}$$

In our discrete setting we used the squared average edge length of the mesh as time step t, as suggested in [8]. We also substitute the identity (I) with a diagonal mass matrix that associates to each vertex in the mesh the sum of the areas/volumes of each incident element (polygon/polyhedron), divided by the number of vertices participating in each such element. Depending on the type of mesh at hand we used different discretizations of the Laplace operator (Δ). For triangle meshes we used the cotangent operator, defined on each vertex v_i as

$$\Delta(v_i) = \sum_{j \in N(i)} (\cot \alpha_{ij} + \cot \beta_{ij})(v_j - v_i) \tag{3}$$

with $N(i)$ being the vertex one ring of v_i, and α_{ij}, β_{ij} the angles opposite to edge (v_i, v_j) [25]. We also used the cotangent Laplacian for tetrahedral meshes [6], which is defined on each vertex v_i as

$$\Delta(v_i) = \sum_{j \in N(i)} \left(\frac{1}{6} \sum_{t \in N(i,j)} |t_{(p,q)}| \cot \phi_{t_{(p,q)}} \right) (v_j - v_i). \tag{4}$$

Again $N(i)$ is the vertex one ring of v_i, and $N(i, j)$ is the fan of tetrahedra having (v_i, v_j) as edge. For each such tetrahedron t, $t(p, q)$ is the edge $(v_p, v_q) \in t$ opposite to (v_i, v_j), and $|t_{(p,q)}|$, $\phi_{t_{(p,q)}}$ are its length and dihedral angle, respectively.

For all the other meshes shown in the paper, we used the combinatorial Laplacian [26]. Notice that different discretizations of the Laplace operator (e.g. [27] for general polygonal meshes) may produce smoother fields, thus having a positive impact on the output result.

Step 2: field initialization. The gradient of the heat flow (∇u) is a vector field that points towards the boundary ζ from any point in the domain (Figure 2c). Our goal is to generate a field that *traverses* ζ in a smooth way. To do so, we generate a vector field X defined as $\nabla u/\|\nabla u\|$ in one region, and $-\nabla u/\|\nabla u\|$ in the other region (Figure 2d). The field will then be smoothed in the subsequent step of the algorithm.

For the computation of the gradient ∇u we use the Green-Gauss method, which provides a sound theoretical basis for gradient computation on any type of discretized closed volume or surface [28]. According to the Green-Gauss method, the average gradient of a scalar function u in a closed volume V can be written as:

$$\overline{\nabla u} = \frac{1}{V} \oiint u\vec{n}\ dA$$

where A is the surface area and $\vec{n}$ the outgoing surface unit normal. We can rewrite the equation for a discrete polyhedral mesh, obtaining:

$$\nabla u = \frac{1}{V} \sum_{f=1}^{|F|} \overline{u}\vec{n}_f A_f$$

Here $|F|$ is the number of faces in the polyhedron, $\overline{u}$ is the arithmetic average of u over the vertices of face f, and $\vec{n}_f$ and A_f are the outgoing unit normal and area of f, respectively. Similarly, the computation of the gradient for a closed surface can be obtained by considering areas instead of volumes, and edge lengths instead of face areas.

Assuming our discrete object is composed of $|P|$ polygons (or polyhedra) and $|V|$ vertices, the gradient operator can be efficiently packed into a $3|P|\times|V|$ matrix G. This representation has a twofold advantage: the first is that by multiplying G for a column vector containing the function values at each vertex in the mesh, a $3|P|$ long column vector containing the serialized gradient can be efficiently computed by matrix vector multiplication; the second is that the transposed matrix $G^\top$ implements the divergence operator, meaning that multiplying $G^\top$ with a vector containing a serialized vector field, gives the divergence of the field. This translates to an extremely compact implementation. In our experiments we used the implementation provided in the CinoLib [29].

Step 3: field smoothing. We smooth the vector field X in order to alleviate the sharp turns induced by the discrete jagged boundary, producing a new copy of it (X'). Specifically, we perform a few iterations of Laplacian smoothing on the dual mesh, meaning that each discrete element X_i takes as vector the average between itself and the vectors associated to its adjacent elements $N(i)$:

$$X'_i = \frac{w_i X_i + \sum_{j\in N(i)} w_j X_j}{\|w_i X_i + \sum_{j\in N(i)} w_j X_j\|} \tag{5}$$

In all our examples we used uniform weights w everywhere. As shown in Figures 1 and 9 this simple smoothing scheme performs well both for regular and irregular tessellations. If necessary, adaptive weighting strategies may be used for uneven tessellations, for example weighting proportionally to areas or volumes. Three to five iterations are usually enough to produce a sufficiently smooth field that crosses the boundary without having sharp turns (Figure 2e). More aggressive smoothing strategies (e.g., more iterations) can be performed to further relax the boundary, allowing it to deviate more from ζ (Figure 8).

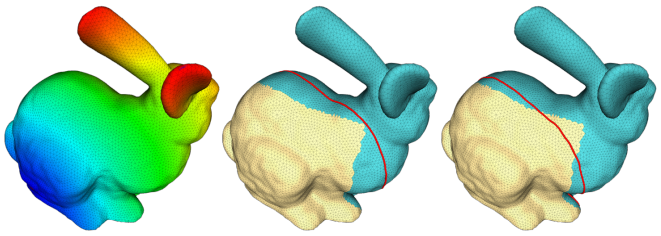

Fig. 3. The function ϕ computed solving Equation 6 (left) and two level sets of it (middle, right). As can be noticed the function does not align with the segmentation boundary. This depends from the fact that smoothing the field X' does not give any guarantee that ϕ will evaluate consistently throughout the whole boundary, especially when aggressive smoothing schemes are applied. In this case we applied 1000 iterations of Laplacian smoothing, introducing a remarkable deviation from the boundary.

Step 4: field integration. The last step consists in generating the function ϕ that aligns to the smoothed vector field X'. The output segmentation boundary will then be a level set of ϕ (Figure 2g). At this point one may think that, similarly to [8], ϕ corresponds to the function that has X' as gradient, which can be computed by solving the Poisson problem

$$\Delta\phi = \nabla X'. \tag{6}$$

However, even though the function traverses ζ in a smooth way, there can be no level set which approximates the discrete boundary well. This is because smoothing the gradient of the function does not give any guarantee on the fact that the function will globally align to the whole boundary ζ. In the general case, it does not (Figure 3). We instead solve for a function ϕ that not only aligns to X', but also has level sets that globally align with ζ. Specifically, we look for the function ϕ that minimizes

$$\arg\min_{\phi} \left\|\Delta\phi - \nabla X'\right\|^2 + \lambda\left\|\phi(\zeta) - \frac{1}{2}\right\|^2, \tag{7}$$

where $\frac{1}{2}$ is the function value that we want to replicate nearby ζ. Notice that $\frac{1}{2}$ provides a reference to the *best fitting* level set, but it could potentially be substituted with any other value. Since we are dealing with the differential properties of the field, the

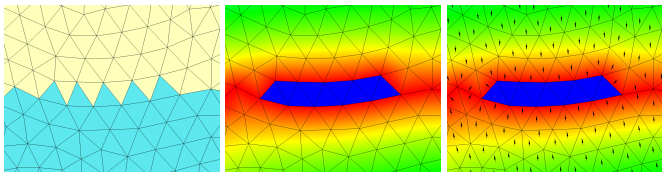

Fig. 4. When a triangle has more than one edge exposed on the segmentation boundary (left), all its three vertices will receive equal heat, leading to a flat function within the triangle (middle) and, thus, an undefined gradient (right). This may lead to unpredictable behaviour during the subsequent smoothing and integration. We avoid these pathological configurations by *padding* the mesh along its boundary, that is splitting elements in order to make sure that each element has at least one non boundary vertex.

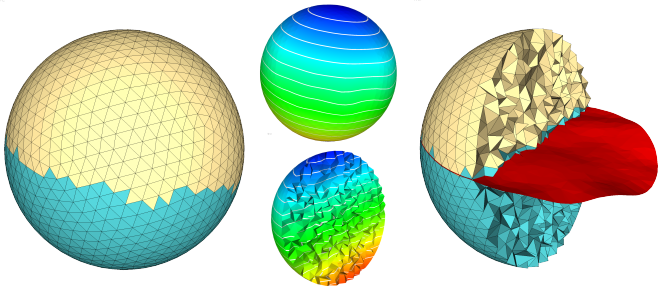

Fig. 5. Smoothly halving a solid sphere. The boundary smoothing algorithm scales on any dimension. Here we consider a bi-partitioned tetrahedral solid sphere (left) and use our continuous function ϕ (middle) to define a smooth cut surface (right).

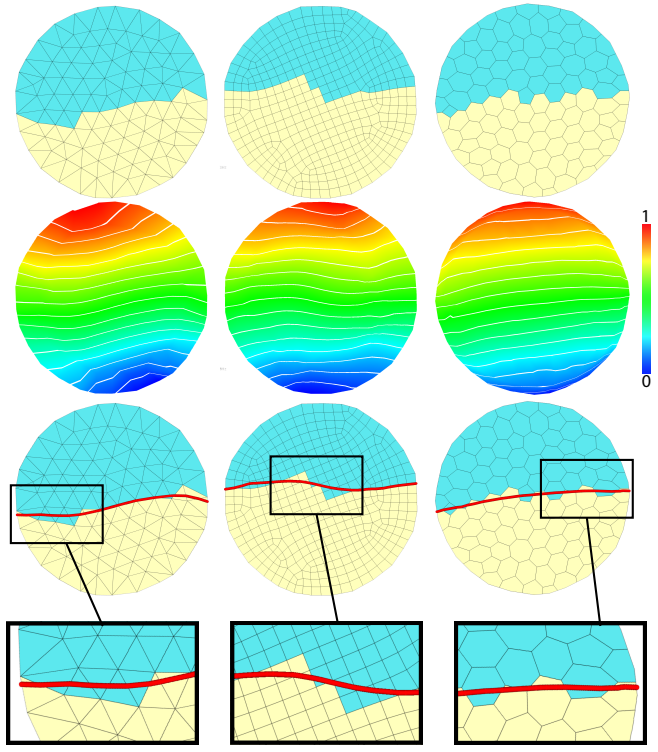

Fig. 6. The algorithm is independent from the discretization. Left to right column: three different segmentations, embedded in the connectivity of a triangular, quadrilateral, and polygonal mesh respectively. Bottom: the best fitting level sets associated to each boundary and relative insets to show small details.

scalar function will shift accordingly. Minimizing equation 7 corresponds to solving a linear system $A\phi = b$, with

$$A = \begin{pmatrix} \Delta \\ M_\zeta \end{pmatrix} \quad b = \begin{pmatrix} \nabla X' \\ \frac{1}{2}^T \end{pmatrix}. \tag{8}$$

Here M_ζ is a sub-matrix having as many rows as the number of vertices in ζ. Each row is null everywhere, and has a single 1 entry corresponding to a vertex in ζ. On the right hand side $1/2^T$ is a column vector containing as many $\frac{1}{2}$ as the number of vertices in ζ. We solve the system using weighted least squares, according to the normal equations $(A^T W A)\phi = (A^T W)b$. The matrix $W = (1\ \lambda)^T$ is diagonal and associates weight 1 for each row corresponding to $\|\Delta\phi - \nabla X'\|^2$ and weight λ for each row corresponding to $\|\phi(\zeta) - 1/2\|^2$. Figures 1, 5, and 6 show some examples of functions ϕ computed with this method.

4. Connectivity issues

The method as presented so far may fail if all the vertices of a discrete element belong to the segmentation boundary. An example is given in Figure 4, where two out of three edges of a triangle are boundary. Notice that the same may happen on volume meshes, e.g. when two out of four faces of a tetrahedron are boundary. These situations are critical because in the first step of the algorithm all boundary vertices receive the same heat. The heat function is therefore flat within the element and its gradient undefined, thus leading to unpredictable results when averaged with its neighbours and finally integrated. The situation can be easily recovered by *padding* the mesh along the segmentation boundary. Padding is a common procedure in mesh optimization. It is used to ensure that each element has at least one unconstrained vertex so as to guarantee enough degrees of freedom for the optimization [30]. Similarly, we aim to ensure that each element has at least one non boundary vertex, so as to avoid flat functions and ill-defined gradients. We split each pathological element in the mesh, adding a new vertex at its barycenter and connecting all its vertices with such vertex. As a result, all the generated sub-elements will have at least one vertex (the newly created one) unconstrained, leading to a non flat function and a well-defined gradient. Notice that this operation can be hidden to the user: the padded mesh is a temporary entity that will be deleted right after computing the smoothed segmentation boundary.

5. Results

We implemented our hyper surface smoothing algorithm as a single threaded C++ application on a MacBook Pro equipped with a 2,9 GHz Intel Core i5 and 16GB of RAM. We used CinoLib [29] for geometry processing and Eigen [31] to solve linear systems. In Figures 1, 2 and 9 we show a variety of results obtained with our implementation.

Performances. From a computational point of view step one (heat flow) and step four (field integration) are the most time consuming steps of the algorithm, and amount to solving one

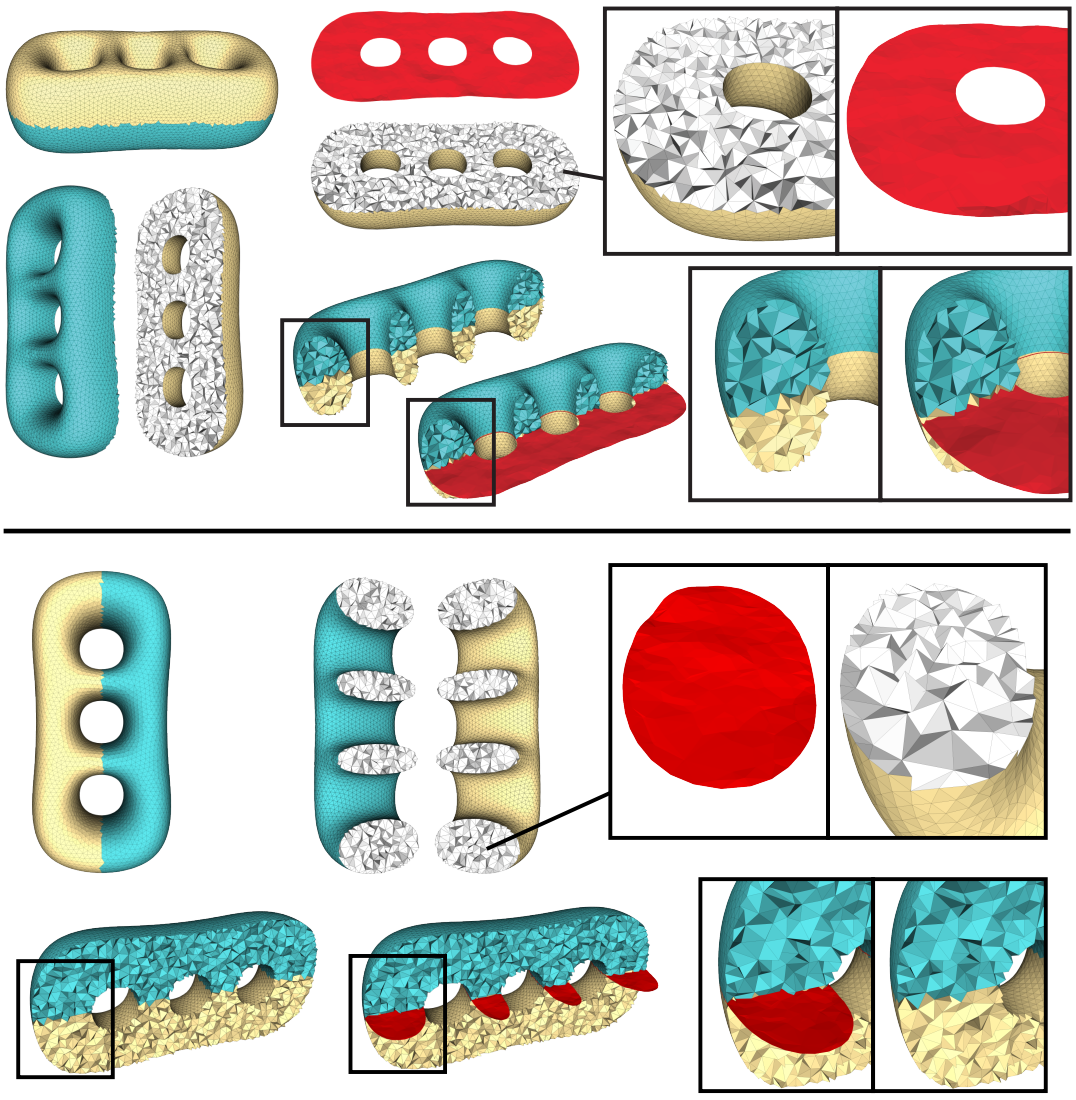

Fig. 7. Two alternative segmentations for the triple torus and relative smoothed boundary. Notice that in both cases the hyper surfaces have non-trivial topology (top: high genus; bottom: multiple connected components). Our method automatically produces level sets that adapt to the topology of the boundary to be approximated.

linear system each. All the other steps (field initialization and smoothing) introduce negligible delays. Detailed time splits for meshes of increasing size are reported in Table 1. As can be noticed, for medium sized meshes the algorithm runs in a fraction of a second and is therefore compatible with interactive use.

Independence from tessellation. We demonstrate independence from the discretization choice in Figure 6, where meshes with different discrete element are processed, ranging from the widespread triangles, to quadrilaterals and general polygons.

Independence from dimension. In Figures 5 and 7 we show three examples of boundary smoothing on three dimensional (i.e. solid) meshes. As previously mentioned the algorithm scales to arbitrarily higher dimensional spaces. To this end, a potential application may be the definition of smooth hypersurfaces that mark the boundaries between different clusters in high dimensional data [32] (e.g. for classification). These clustering problems often arise in data mining and machine learning. An interesting property of our method is that the differential operators we rely on are purely intrinsic [33], meaning that the computational effort depends only on the dimension of the data itself, and not the dimension of the space in which data is embedded.

Topology control. In Figure 7 we show two examples of solid bi-partitions where the boundary between the two regions has non trivial topology (e.g., high genus, multiple connected components). These cases do not require special handling: our method automatically produces level sets that adapt to the topology of the boundary to be relaxed. Holes and disconnected components naturally arise wherever necessary. Notice that the algorithm does not allow the user to explicitly control the topology of the level sets, and no guarantees can be provided in this sense. However, from our experiments this seems to be more a

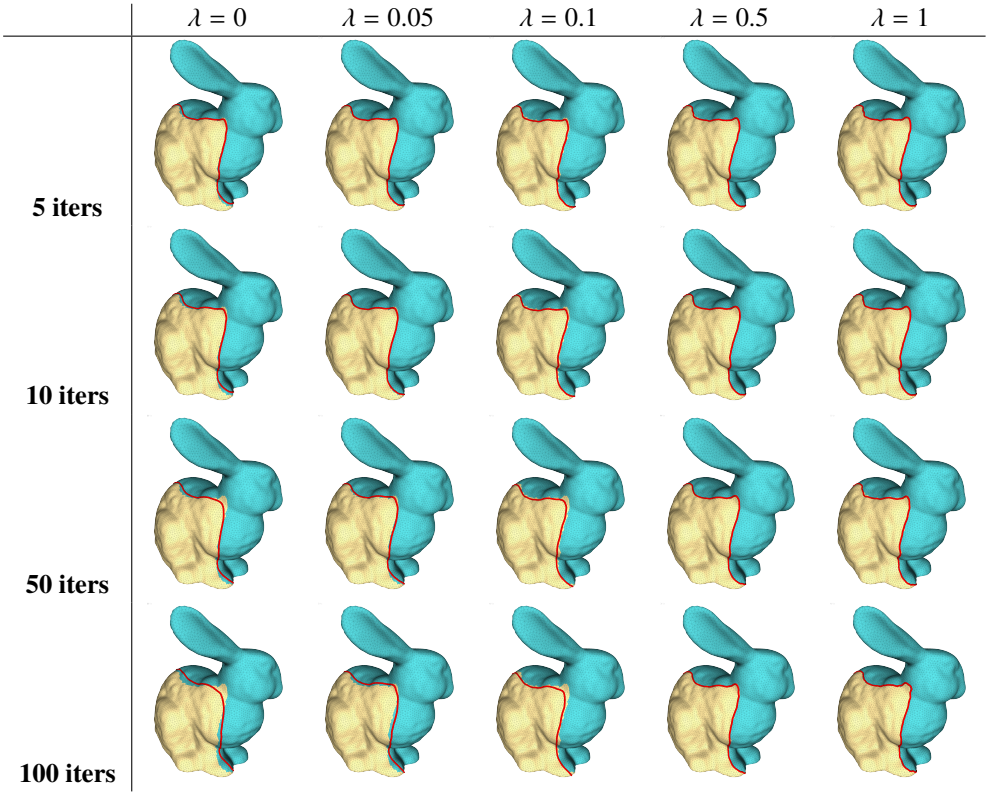

Fig. 8. Combined study of the two parameters from which the algorithm depends: the number of smoothing iterations (rows) and λ, which promotes consistent function evaluation throughout the original jagged boundary (columns). As can be noticed the algorithm is pretty stable. For $\lambda \geq 0.5$ (two rightmost columns) the method consistently offers a faithful approximation of the original boundary, regardless the number of smoothing iterations. To reduce fidelity and promote smoothness, the user can choose $\lambda < 0.5$ and progressively increase the smoothing iterations (bottom left part of the grid).

theoretical than a practical limitation: we never observed differences in topology between a discrete boundary and its smoothed counterpart.

Parameter tuning. The algorithm depends from two parameters: the number of smoothing iterations of the field X, and the scalar λ, which balances between field alignment and consistent evaluation of the function ϕ throughout the discrete boundary. In Figure 8 we show a study of how different combinations of these two parameters affect our boundary relaxation scheme. The method is easy to control: for $\lambda \geq 0.5$ it consistently offers a faithful approximation of the original boundary, regardless the number of smoothing iterations. Progressively smoother versions of the boundary, with increasing deviation from the original one, can be produced using $\lambda < 0.5$ and growing the number of smoothing iterations.

Implementation. Being based on widespread discrete differential operators included in many freely available geometry processing libraries, the algorithm is quite easy to implement. If differential operators are overloaded to work with different discrete entities while maintaining the same C++ interface, the algorithm can be implemented only once and used with any discrete object. We exploited general programming to produce a unique implementation that we used to produce all the results shown in this article, including triangle, quadrilateral and polygonal meshes as well as polyhedral meshes. We publicly release our implementation at the following address: `https://github.com/maxicino/HyperSurfaceSmoothing`.

Num. verts	Heat Flow (s)	Field creation (s)	Smoothing (s)	Integration (s)	TOT (s)
1K	6×10^{-3}	4×10^{-5}	3×10^{-3}	9×10^{-3}	16×10^{-3}
4K	2×10^{-2}	1×10^{-4}	9×10^{-4}	5×10^{-2}	72×10^{-3}
15K	9×10^{-2}	2×10^{-4}	2×10^{-3}	0.37	0.46
60K	0.57	1×10^{-3}	1×10^{-2}	3.7	4.3
250K	3.62	4×10^{-3}	5×10^{-2}	27.5	31.2
1M	29.6	1×10^{-2}	0.19	295.3	325.1

Table 1. Time performances of the boundary smoothing method, tested on a mesh containing 1K vertices, and progressively refined up to 1M vertices. The most time consuming parts are the computation of the heat flow and and the vector field integration, which involve the resolution of a linear system each. The generation of the field and its smoothing (always 3 iterations for this experiment) introduce negligible overhead. We used the built-in LL^T solver provided by Eigen for linear systems. Better performances could be obtained by adopting faster (possibly parallel) solvers.

6. Conclusions and Future Works

We introduced a novel smoothing operator to relax the boundaries of discrete segmentations. The operator is inspired by the heat geodesic method [8], from which it inherits the ability to scale on different discrete domain representations and to generalize to high dimensional spaces.

We experimentally validated our method on a variety of discrete entities, ranging from triangle/quad/polygon meshes to

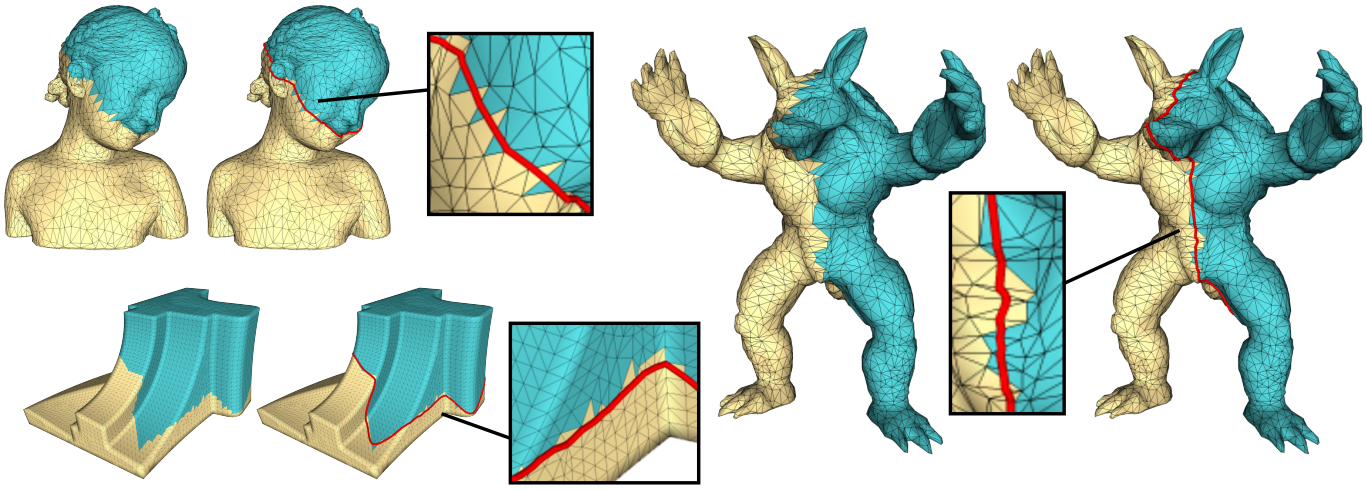

Fig. 9. A gallery of results produced using our method.

tetrahedral meshes. We also studied the influence of the parameters that control the algorithm, showing its stability and ease to control.

Limitations and future works. At the moment this method can be used to relax discrete boundaries shared between *pairs* of regions. Indeed, one interesting question that we plan to investigate further is how to extend it to more complex scenarios, where more than two regions are involved at the same time. We observe that when more than two regions are involved the boundary not only contains $(n-1)$ dimensional components, but also $(n-2)$ dimensional ones. As a simple example one may think of three boundary curves meeting at a common point, or three boundary surfaces meeting at a common curve (see e.g. Figure 6 in [6]). How to globally smooth such complex boundaries assembly remains an open challenge that we wish to tackle in future works.

As suggested by one of the reviewers of the preliminar version of this work [9], it would be interesting to edit the generation of the function ϕ in order to finely control the geometry of the cut. A candidate application could be 3D printing, where big objects need be split in order to fit the printing chamber, and the cut must be computed so as not to generate overhangs with respect to the building direction [34]. The suitability of our framework to solve problems of this kind is currently under investigation. A direction that seems promising is the control of the local surface orientation at integration time, obtained by restricting the angle between the printing direction and the function gradient.

Acknowledgements

This work is supported by the EU ERC Advanced Grant CHANGE, grant agreement No.694515. The author wishes to thank all the anonymous reviewers of this work for fruitful suggestions on how to improve it, and also for pointing me to interesting outcomes of this research related to fabrication.

References

[1] Boykov, Y, Veksler, O, Zabih, R. Fast approximate energy minimization via graph cuts. IEEE Transactions on pattern analysis and machine intelligence 2001;23(11):1222–1239.

[2] Livesu, M, Vining, N, Sheffer, A, Gregson, J, Scateni, R. Polycut: Monotone graph-cuts for polycube base-complex construction. ACM Transactions on Graphics (Proc SIGGRAPH ASIA 2013) 2013;32(6). doi:10.1145/2508363.2508388.

[3] Kalogerakis, E, Hertzmann, A, Singh, K. Learning 3d mesh segmentation and labeling. In: ACM Transactions on Graphics (TOG); vol. 29. ACM; 2010, p. 102.

[4] Reuter, M, Biasotti, S, Giorgi, D, Patanè, G, Spagnuolo, M. Discrete laplace–beltrami operators for shape analysis and segmentation. Computers & Graphics 2009;33(3):381–390.

[5] Mortara, M, Patanè, G, Spagnuolo, M, Falcidieno, B, Rossignac, J. Plumber: a method for a multi-scale decomposition of 3d shapes into tubular primitives and bodies. In: Proceedings of the ninth ACM symposium on Solid modeling and applications. Eurographics Association; 2004, p. 339–344.

[6] Livesu, M, Attene, M, Patané, G, Spagnuolo, M. Explicit cylindrical maps for general tubular shapes. Computer-Aided Design 2017;doi:10.1016/j.cad.2017.05.002.

[7] Attene, M, Mortara, M, Spagnuolo, M, Falcidieno, B. Hierarchical convex approximation of 3d shapes for fast region selection. In: Computer graphics forum; vol. 27. Wiley Online Library; 2008, p. 1323–1332.

[8] Crane, K, Weischedel, C, Wardetzky, M. Geodesics in heat: A new approach to computing distance based on heat flow. ACM Transactions on Graphics (TOG) 2013;32(5):152.

[9] Livesu, M. Heat Flow Based Relaxation of n Dimensional Discrete Hyper Surfaces. In: Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference. The Eurographics Association. ISBN 978-3-03868-048-2; 2017,doi:10.2312/stag.20171222.

[10] Shamir, A. A survey on mesh segmentation techniques. In: Computer graphics forum; vol. 27. Wiley Online Library; 2008, p. 1539–1556.

[11] Au, OKC, Zheng, Y, Chen, M, Xu, P, Tai, CL. Mesh segmentation with concavity-aware fields. IEEE Transactions on Visualization and Computer Graphics 2012;18(7):1125–1134.

[12] Ji, Z, Liu, L, Chen, Z, Wang, G. Easy mesh cutting. In: Computer Graphics Forum; vol. 25. Wiley Online Library; 2006, p. 283–291.

[13] Lee, Y, Lee, S, Shamir, A, Cohen-Or, D, Seidel, HP. Intelligent mesh scissoring using 3d snakes. In: Computer Graphics and Applications, 2004. PG 2004. Proceedings. 12th Pacific Conference on. IEEE; 2004, p. 279–287.

[14] Kaplansky, L, Tal, A. Mesh segmentation refinement. In: Computer Graphics Forum; vol. 28. Wiley Online Library; 2009, p. 1995–2003.

[15] Panozzo, D, Baran, I, Diamanti, O, Sorkine-Hornung, O. Weighted averages on surfaces. ACM Transactions on Graphics (TOG) 2013;32(4):60.

[16] Lawonn, K, Gasteiger, R, Rössl, C, Preim, B. Adaptive and robust curve smoothing on surface meshes. Computers & Graphics 2014;40:22–35.
[17] Lee, Y, Lee, S. Geometric snakes for triangular meshes. In: Computer Graphics Forum; vol. 21. Wiley Online Library; 2002, p. 229–238.
[18] Kass, M, Witkin, A, Terzopoulos, D. Snakes: Active contour models. International journal of computer vision 1988;1(4):321–331.
[19] Benjamin, W, Polk, AW, Vishwanathan, S, Ramani, K. Heat walk: Robust salient segmentation of non-rigid shapes. In: Computer Graphics Forum; vol. 30. Wiley Online Library; 2011, p. 2097–2106.
[20] Belyaev, AG, Fayolle, PA. On variational and pde-based distance function approximations. In: Computer Graphics Forum; vol. 34. Wiley Online Library; 2015, p. 104–118.
[21] Herholz, P, Haase, F, Alexa, M. Diffusion diagrams: Voronoi cells and centroids from diffusion. In: Computer Graphics Forum; vol. 36. Wiley Online Library; 2017, p. 163–175.
[22] Bronstein, AM, Bronstein, MM, Kimmel, R, Mahmoudi, M, Sapiro, G. A gromov-hausdorff framework with diffusion geometry for topologically-robust non-rigid shape matching. International Journal of Computer Vision 2010;89(2):266–286.
[23] Patané, G. Star-laplacian spectral kernels and distances for geometry processing and shape analysis. In: Computer Graphics Forum; vol. 35. Wiley Online Library; 2016, p. 599–624.
[24] Desbrun, M, Meyer, M, Schröder, P, Barr, AH. Implicit fairing of irregular meshes using diffusion and curvature flow. In: Proceedings of the 26th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co.; 1999, p. 317–324.
[25] Meyer, M, Desbrun, M, Schröder, P, Barr, AH. Discrete differential-geometry operators for triangulated 2-manifolds. In: Visualization and mathematics III. Springer; 2003, p. 35–57.
[26] Xu, Y, Chen, R, Gotsman, C, Liu, L. Embedding a triangular graph within a given boundary. Computer Aided Geometric Design 2011;28(6):349–356.
[27] Alexa, M, Wardetzky, M. Discrete laplacians on general polygonal meshes. In: ACM Transactions on Graphics (TOG); vol. 30. ACM; 2011, p. 102.
[28] Sozer, E, Brehm, C, Kiris, CC. Gradient calculation methods on arbitrary polyhedral unstructured meshes for cell-centered cfd solvers. In: Proceedings of the 52nd Aerospace Sciences Meeting, National Harbor, MD, USA; vol. 1317. 2014,.
[29] Livesu, M. cinolib: A generic programming C++ library for processing polygonal and polyhedral meshes. 2017. Https://github.com/maxicino/cinolib/.
[30] Livesu, M, Muntoni, A, Puppo, E, Scateni, R. Skeleton-driven adaptive hexahedral meshing of tubular shapes. Computer Graphics Forum 2016;35(7):237–246. doi:10.1111/cgf.13021.
[31] Guennebaud, G, Jacob, B, et al. Eigen v3. http://eigen.tuxfamily.org; 2010.
[32] Assent, I. Clustering high dimensional data. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2012;2(4):340–350.
[33] Boscaini, D, Eynard, D, Bronstein, MM. Shape-from-intrinsic operator. arXiv preprint arXiv:14061925 2014;.
[34] Livesu, M, Ellero, S, Martínez, J, Sylvain, L, Attene, M. From 3d models to 3d prints: an overview of the processing pipeline. Computer Graphics Forum 2017;36(2):537–564. doi:10.1111/cgf.13147.