Making Contour Trees Subdomain-Aware

Nathanael Berglund* and Andrzej Szymczak†

Abstract

We describe a simple and efficient algorithm for computing a variant of a contour tree that describes, for each contour c, the number of connected components in the intersection of c with a fixed simply connected subdomain $\mathcal{S}$. The algorithm requires $O(n + t \log t)$ time, where n is the size of the input mesh and t is the total number of critical points of the scalar field $\mathcal{F}$ and of the restriction of $\mathcal{F}$ to $\mathcal{S}$. We show how to use our algorithm to label the edges of the contour tree of a 3D scalar field with complete information on the topology of the corresponding contours in $O(n + t \log t)$ time.

1 Introduction

Contour trees (CTs) are considered an important tool allowing one to concisely describe the structure of isosurfaces in volume data as well as the way they evolve and interact as the isovalue is varied. A CT can be defined as a quotient space $\mathcal{D}/\equiv$ where $\mathcal{D}$ is the domain of a scalar field $\mathcal{F}$ and, for $x, y \in \mathcal{D}$, $x \equiv y$ if and only if x and y belong to the same *contour*, i.e. a connected component of a set of the form $\mathcal{F}^{-1}(c)$ for some scalar c. A scalar field is typically represented as a simplicial complex with values at vertices or a regular (rectilinear) grid of samples. Linear or multilinear interpolation is used to obtain values at points other than the samples. Most scalar fields that appear in applications are defined on simply connected domains. In this case the CT is indeed a tree.

Contour trees been used as a tool to enhance scalar field visualization [1], speed up certain types of queries in geographical information systems [2] and facilitate isosurface extraction from volume datasets by helping to compute small seed sets [5, 6]. These applications motivated efforts to develop increasingly simpler, faster and more general algorithms for computing contour trees. An $O(n \log n)$ algorithm for computing the contour tree in two dimensions was given in [2]. A simpler version of the 2D algorithm and an $O(n^2)$ algorithm for higher dimensions is given in [5]. An $O(n \log n)$ algorithm that works in three dimensions was proposed in [9] and subsequently simplified and generalized to any dimension in [3]. An $O(n + t \log t)$ implementation of this algorithm is described in [4]. The work [8] describes a method for labeling the edges of the contour tree with Betti numbers in $O(n + t \log n)$ time where n is the size of the mesh and t is the number of critical points.

* School of Math., Georgia Tech, berglund@math.gatech.edu
† College of Computing, Georgia Tech, andrzej@cc.gatech.edu

The original motivation for this paper was to improve the algorithm for labeling contour tree edges with Betti numbers of their associated contours introduced in [8] to give complete topological information about each contour, and to do so without increasing the asymptotic running time. Contours in 3-dimensional scalar fields are orientable 2-manifolds with boundary. The well-known classification theorem for 2-manifolds [7] states that any orientable two dimensional manifold is homeomorphic to the sphere or to a connected sum of some number of two-dimensional tori. An orientable 2-manifold with boundary is homeomorphic to the sphere or a connected sum of tori with some number of disjoint topological disks removed, thus the boundary is composed of a number of disjoint "loops". The Betti numbers do not provide enough information to discriminate the topology of a connected 2-manifold with boundary. For a connected surface of genus g with k disjoint disks removed, its Betti numbers are given by:

$$\beta_0 = 1,\ \beta_1 = \begin{cases} 2g & \text{if } k \leq 1 \\ 2g + k - 1 & \text{if } k > 1 \end{cases},\ \beta_2 = \begin{cases} 0 & \text{if } k \neq 0 \\ 1 & \text{if } k = 0 \end{cases}$$

In particular, this means that it is impossible to distinguish a torus with one disk removed from a double torus with three disks removed by looking at the Betti numbers alone. The topology of a surface with a boundary can be uniquely determined if one knows both the number, k, of its boundary loops, and the Euler characteristic $\chi = \beta_2 - \beta_1 + \beta_0$. A special case of the algorithm discussed in this paper provides an efficient way to label the edges of the contour tree with the number of boundary loops of the corresponding contours. Together with a method of labeling the edges of the contour tree with the Euler characteristic introduced in [8] one can label the edges of the contour tree with numbers providing a complete description of the corresponding contour topology. Our algorithm requires $O(n + t \log t)$ time and is applicable to scalar fields represented by both structured and unstructured meshes. It takes advantage of the relationship between two contour trees:

1. The restricted contour tree (denoted by $\mathcal{T}_{\text{restricted}}$): contour tree for the scalar field $\mathcal{F}$ restricted to the given subdomain $\mathcal{S}$.

2. The full contour tree (denoted by $\mathcal{T}_{\text{full}}$): contour tree for the entire mesh $\mathcal{C}$ augmented with the critical points of $\mathcal{F}$ restricted to $\mathcal{S}$.

The dominating cost is the cost of computing these two trees. Once they are computed, the number of connected "subcontours" of each contour when it is restricted to $\mathcal{S}$ can be assigned to the edges of $\mathcal{T}_{\text{full}}$ in time linear to its size.

Anticipating other applications of this technique, we describe it as a general method of making each contour c, represented as an edge in $\mathcal{T}_{\text{full}}$, aware of the number of connected components of $c \cap \mathcal{S}$. For example, one may be interested in finding contours that intersect a given cross-section through a volume dataset at two closed curves while intersecting another cross-section at one closed curve. Our algorithm can be used to provide information about isovalues that would lead to these types of contours. We define a subdomain-aware contour tree as a contour tree $\mathcal{T}$ each of whose edges e is each labeled with the number of connected components of $c_e \cap \mathcal{S}$ for a subdomain $\mathcal{S}$ (where c_e denotes a contour represented the edge e).

2 Example

Before we proceed to formal description of our algorithm, we illustrate the underlying idea with a simple 2D example. Consider a height field whose contour plot is shown in Figure 1. Recall that by a critical point we mean a point where the local structure of contours changes. The most obvious critical points are local maxima and local minima. This is where contours appear and disappear (respectively) as the isovalue is decreased. Apart from local extrema, there are critical points at which contours merge, split or change topology. For example, at B the contour which appears at G changes topology (from topological loop to a line segment). At H it merges with the contour that appears at I. As the isovalue is decreased, this contour hits the boundary of the dataset at F and is split into two contours, one of them disappearing at A shortly after the split and the other one undergoing more complex evolution. (Figure 1). If necessary, the contour tree can be augmented with extra vertices. In our case, we will use a contour tree augmented with critical points of $\mathcal{F}$ restricted to $\mathcal{S}$ (the union of the three intervals AO, EQ, and LN, shown as dashed lines in Figure 1). In our case, there are three critical points (L, M, and N) of the restricted scalar field that are regular (i.e. not critical) points in the full dataset. M is a local maximum of the restricted scalar field. At L and N, two contours in the restriction (one arriving from above, one along the horizontal interval) merge to form a contour moving down along either the left or right edge of the dataset. By the height of a vertex of a contour tree we shall mean the scalar value of the corresponding point in the scalar field.

The contour trees $\mathcal{T}_{\text{full}}$ and $\mathcal{T}_{\text{restricted}}$ for the height field $\mathcal{F}$ and subdomain $\mathcal{S}$ as shown in Figure 1 are shown in Figure 2. Consider an edge e in the $\mathcal{T}_{\text{restricted}}$. Because $\mathcal{T}_{\text{full}}$ contains all critical points of the restricted height field, endpoints of e are vertices of $\mathcal{T}_{\text{full}}$. There is a unique shortest path joining the two vertices. In fact, this path is monotonic (i.e. visits vertices of $\mathcal{T}_{\text{full}}$ in the order of either increasing or decreasing scalar value). This is particularly easy to see if the contour trees are thought of as quotient spaces. Then, e

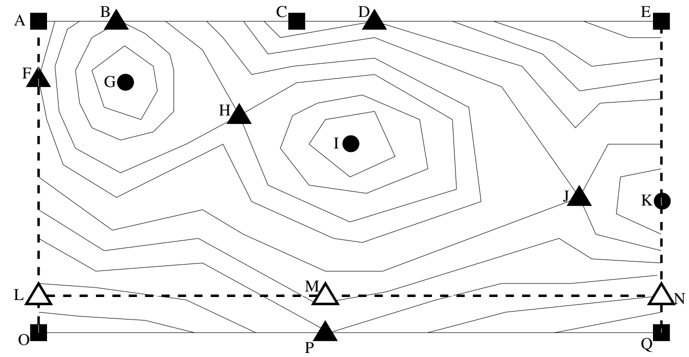

Figure 1: Contour plot of the example height field. Local maxima are shown as black circles, local minima - as black squares. Triangles indicate other types of critical points. Points that are regular but are critical in the restriction of the height field to the subdomain (indicated by dashed lines, the union of the left and right edges of the domain and the line LN connecting the two) are shown as hollow triangles (L, M, and N).

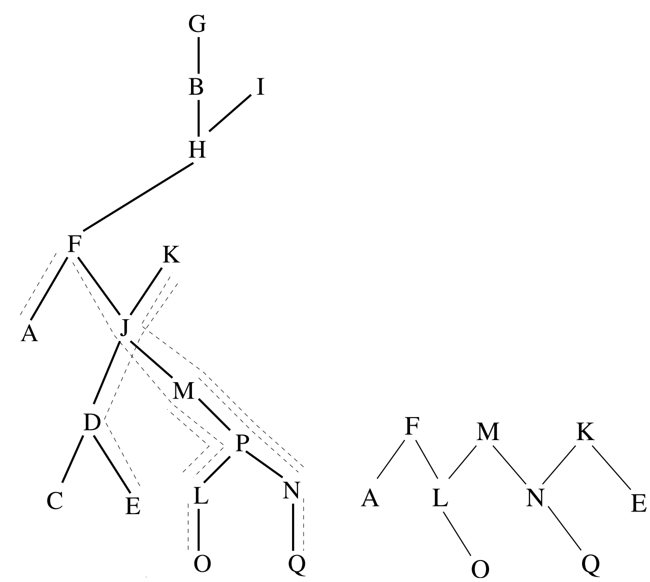

Figure 2: Left: contour tree (its edges are shown as solid lines) of the height field shown in Figure 1. Right: contour tree of the height field shown in Figure 1 restricted to the subdomain.

can be viewed as a continuous increasing path in $\mathcal{T}_{\text{restricted}}$. The inclusion map from $\mathcal{S}$ into $\mathcal{C}$ yields a continuous map $\iota : \mathcal{T}_{\text{restricted}} \to \mathcal{T}_{\text{full}}$. Applying this map to the path induces an increasing path in $\mathcal{T}_{\text{full}}$, whose endpoints are the same as endpoints of e.

In Figure 2, the paths corresponding to edges of $\mathcal{T}_{\text{restricted}}$ are shown in dashed lines (henceforth referred to as "subcontour paths"). Clearly, given a contour c, the number of connected components of $c \cap \mathcal{S}$ is equal to the size of the pre-image of the corresponding point in $\mathcal{T}_{\text{full}}$ under the map ι. This number is the same for all contours corresponding to an edge e of $\mathcal{T}_{\text{full}}$. It can be computed as the number of times e is traversed by a subcontour path. The simplest way to

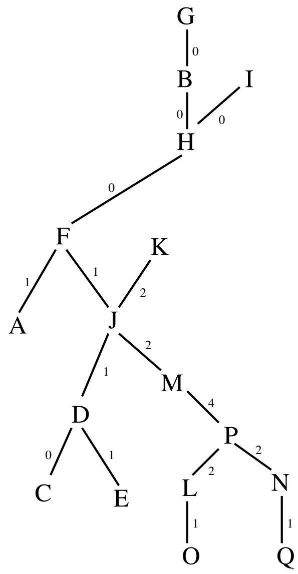

Figure 3: Subdomain-aware version of the contour tree in Figure 2.

compute these numbers would be to walk each subcontour path and increment a counter associated with each edge of $\mathcal{T}_{\text{full}}$ each time it is traversed. However, this can potentially result in quadratic complexity. Therefore, we instead use a method similar to [8]: Notice that for any vertex v of $\mathcal{T}_{\text{full}}$ we have the following relation involving the labels on the edges out of v: the difference between the total number of times all edges going down from v are traversed by subcontour paths and the total number of times all edges going up from v are traversed by subcontour paths is equal to either zero if v is not present in $\mathcal{T}_{\text{restricted}}$ (e.g. J or P in Figure 2) or the difference of the number of edges going down from v and the number of edges going up from v in $\mathcal{T}_{\text{restricted}}$ (e.g. M or K). This allows one to compute the edge labels in linear time by greedily selecting and solving one with just one unknown variable. In our example, the labels for edges of the contour tree (shown in Figure 3) can, for example, be computed for the edges in the following order: BG, HB, HI, FH, AF, JF, JK, CD, ED, DJ, MJ, PM, OL, LP, PN, NQ.

3 Summary of the Algorithm

The input to our algorithm consists of:

1. A piecewise linear scalar field $\mathcal{F}$ specified as a simply connected simplicial complex $\mathcal{C}$ with scalar values at vertices

2. A subdomain, a simply connected subcomplex $\mathcal{S}$ of $\mathcal{C}$.

As the output, we produce the *subdomain-aware* contour tree $\mathcal{T}$, which is a contour tree for the input scalar field with non-negative integer labels assigned to edges. The label of the edge e is the number of connected components of $c_e \cap \mathcal{S}$ (where c_e is the contour represented by e). The vertices of $\mathcal{T}$ are the critical points of $\mathcal{F}$ and the critical points of $\mathcal{F}$ restricted to $\mathcal{S}$.

Critical points can be defined as vertices of the domain for which the lower or upper link is either empty or has more than two connected components. The upper (respectively lower) link consists of all simplices (of any dimension) whose vertices are all adjacent to v and have values greater (resp. lower) than the value at v. All vertices that are not critical are called regular.

Our algorithm first computes $\mathcal{T}_{\text{full}}$ and $\mathcal{T}_{\text{restricted}}$, and then uses the structure of these two trees to find the edge labels on $\mathcal{T}$.

3.1 Contour trees

The contour trees $\mathcal{T}_{\text{full}}$ and $\mathcal{T}_{\text{restricted}}$ are computed using the algorithm of [4] in $O(n+t\log t)$ time, where t is the number of vertices in the output trees. In our case, apart from critical points of $\mathcal{F}$, the tree $\mathcal{T}_{\text{full}}$ must also contain the critical points of $\mathcal{F}$ restricted to $\mathcal{S}$ (some of them may be regular relative to the full domain, e.g. L,M and N in Figure 1).

3.2 Edge labels

Now we proceed to computing the edge labels. For each vertex v of the tree $\mathcal{T}_{\text{full}}$ let $U(v)$ (respectively, $L(v)$) be the set of vertices adjacent to v with larger (respectively, smaller) height. For a vertex v of $\mathcal{T}_{\text{restricted}}$, denote by $d^+(v)$ (respectively, $d^-(v)$) the number of vertices adjacent to v in $\mathcal{T}_{\text{restricted}}$ with larger (respectively, smaller) height. Finally, for a vertex v of $\mathcal{T}_{\text{full}}$ let $\Delta(v)$ be zero if v is not a vertex of the $\mathcal{T}_{\text{restricted}}$ and $d^+(v) - d^-(v)$ otherwise. An argument outlined in the previous section shows that for each vertex v of $\mathcal{T}_{\text{full}}$ the following equation holds (n_e is the edge label for an edge e):

$$\sum_{w\in U(v)} n_{\{v,w\}} - \sum_{u\in L(v)} n_{\{v,u\}} = \Delta v. \tag{1}$$

This is a system of linear equations, in which the edge labels are unknowns, with the same structure as equations for Euler characteristic of contours discussed in [8] and it can be solved in the same way. The idea is to solve the equations in an order which ensures that there is only one unknown with undetermined value in each equation being solved at any time. We maintain a tree $\mathcal{T}_{\text{unlabeled}}$, which is the subtree of $\mathcal{T}_{\text{full}}$ whose edges have not yet been labeled, and for each vertex v of $\mathcal{T}_{\text{unlabeled}}$, we store a number `delta(v)`, defined by the left-hand side of equation (1), where $U(v)$ and $L(v)$ are interpreted in the sense of $\mathcal{T}_{\text{unlabeled}}$. When $\mathcal{T}_{\text{unlabeled}} = \mathcal{T}_{\text{full}}$, we have `delta(v)` $= \Delta v$. We also use a queue of leaf edges of $\mathcal{T}_{\text{unlabeled}}$ as an auxiliary datastructure. Initially, $\mathcal{T}_{\text{unlabeled}}$ contains all edges of $\mathcal{T}_{\text{full}}$ and the queue contains all its leaf edges. In a loop, we take an edge e out of the queue and solve its equation (which will already be solved up to a $\pm$ sign by the time e enters the queue) to obtain the label for e, and then remove e from $\mathcal{T}_{\mathrm{unlabeled}}$, updating `delta(v)` for the vertex v of e that remains. If, as a result of that removal, v becomes a leaf vertex, we insert its corresponding leaf edge into the queue. The whole process terminates when the queue becomes empty and labels of all edges of $\mathcal{T}_{\mathrm{full}}$ are known. Clearly, the whole process takes $O(t)$ time.

4 Application: full description of contour topology for 3D scalar fields

By applying our algorithm to a scalar field defined on a simply connected tetrahedral mesh embedded in the 3-dimensional space, with the subdomain being the boundary of the domain, one can label edges of the contour tree with the numbers of the boundary loops of the corresponding contours. The algorithm of [8] can be used to label the edges of the same tree with the Euler characteristic of the contours. Both labels provide complete information about the contour topology. The contour tree together with both labels can be computed (for general tetrahedral meshes) in $O(n + t \log t)$ time, where t is the size of the output tree (equal to the number of vertices of the input mesh that are either critical points relative to the whole domain or its boundary). Thus, by using the algorithm of [4] to compute the initial contour trees, we improve the running time of [8], and also provide a complete description of contour topology in three dimensions, while preserving the aymptotic running time of $O(n + t \log t)$.

5 Summary

We have described a simple and efficient algorithm that combines prior algorithms for computing contour trees and uses a similar technique to that in [8] to label the edges of a contour tree, in order to answer a question that, to our knowledge, has not previously been answered: "Can one efficiently compute the complete topological information for each contour of a contour tree representing a dataset in $\mathbb{R}^3$?" We have shown that this can indeed be done in $O(n + t \log t)$ time. Unfortunately in higher dimensions however, a complete classification of all topological manifolds is yet to be found. We have also noticed that our algorithm naturally generalizes to computing the number of connected components of the intersection of each contour c with a fixed simply connected subdomain $\mathcal{S}$, and can be implemented in any dimension on any type of mesh, a direct result of the fact that the existing algorithm [4] for computing the contour tree can do so, and that our algorithm relies solely upon having a pair of input contour trees, not on the original data set. We believe this generalized algorithm may find other applications besides our original goal.

6 Acknowledgements

The second author was supported by NSF/CARGO grant 0138420.

References

[1] C. Bajaj, V. Pascucci and D. R. Schikore *The Contour Spectrum*, In Proc. IEEE Visualization, pp 167-175, 1997.

[2] M. de Berg and M. van Kreveld, *Trekking in the Alps without freezing or getting tired*, Algorithmica, 18 (1997), pp.306–323.

[3] H. Carr and J. Snoeyink and U. Axen, *Computing contour trees in all dimensions*, Computational Geometry 24(2003), pp. 75–94.

[4] Y. Chiang and X. Lu, *Simple and Optimal Output-Sensitive Computation of Contour Trees*, Tech Report TR-CIS-2003-02, Polytechnic University, June 2003.

[5] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci , and D. R. Schikore, *Contour trees and small seed sets for isosurface traversal*, In Proc. 13th ACM Ann. Sympos. on Comp. Geom. (SoCG),1997, pp. 212-220.

[6] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci and D. Schikore *Contour Trees and Small Seed Sets for Isosurface Generation*, in: Topological Data Structures for Surfaces, Sanjay Rana (Editor), Wiley Europe, March, 2004.

[7] W. S. Massey, *Algebraic Topology: An Introduction*, Graduate Texts in Mathematics No. 127, Springer-Verlag, 1987.

[8] V. Pascucci and K. Cole-McLaughlin, *Efficient Computation of the Topology of Level Set.*, Proc. IEEE Visualization 2002, Boston MA, October 2002, pp. 187–194.

[9] S. P. Tarasov and M. N. Vyalyi, *Construction of contour trees in 3D in $O(n \log n)$ steps*, Proc. 14th ACM Ann. Sympos. Comput. Geom. 1998, pp. 68–75.