Efficient Contour Tree Construction and Computation of Betti Numbers in Scalar Fields

Tobias Lenz* Günter Rote*

Submitted to the *19th European Workshop on Computational Geometry, 2003*

Abstract

A new algorithm to construct contour trees is introduced which improves the runtime of known approaches. It also generates additional topological information about the data which can be used to compute the Betti numbers for all possible level sets.

1 Introduction

Visualizing Contours in Scalar Fields. A commonly used technique to store large amounts of data is in a scalar field. This data could be measurement results in any dimension, e.g. elevation information in geographic information systems. The visualization of the data is usually done by drawing level sets which are points of equal value. A connected component in a level set is called *contour*. In two dimensions the scalar value can be interpreted as the height over a two-dimensional domain. In general the domain should be given as a simplicial complex and the scalar values are interpolated over discreet values at the vertices.

The Contour Tree. To grant high speed for interactive systems drawing contours, an efficient structure is needed. Quickly drawing a level set includes the knowledge of how many connected components the system has to draw and where they are. It is sufficient then to have a single simplex through which the requested contour passes, a so called *seed*. By recursively checking the simplices in its neighborhood the whole connected component of the level set can be traced.

The contour tree is helpful for creating a sufficient set of seeds. Every node in the contour tree represents a point at a level where the number of components changes. Every edge in the contour tree spans the interval between the values of the represented points of the two incident nodes and represents a connected component. This works for any dimension.

Betti Numbers. In some applications topological information about the surface is needed, e.g. the Betti numbers. In topology the i-th Betti number β_i is defined as the rank of the i-dimensional homology group. For a d-dimensional object the Betti numbers $\beta_d, \beta_{d+1}, \ldots$ are all zero. In this paper the Betti numbers are only computed for up to three dimensions, therefore we only care about $\beta_0, \beta_1, \beta_2$. They have a very graphic meaning: β_0 is the number of connected components, β_1 is the number of tunnels through them and β_2 is the number of enclosed voids.

Previous Work. Carr et al. [1] presented a fast sweep algorithm. They do two simple sweeps over the data in order of increasing and decreasing scalar values to create two trees. In a third step these trees are combined to the final contour tree. The required time is $O(N + n \log n)$ for any dimension where N denotes the size of the simplicial complex and n the number of vertices.

Pascucci [2] augmented the contour tree for three

*Institut für Informatik, Freie Universität Berlin, Takustraße 9, 12247 Berlin, Germany,
tlenz, rote@inf.fu-berlin.de

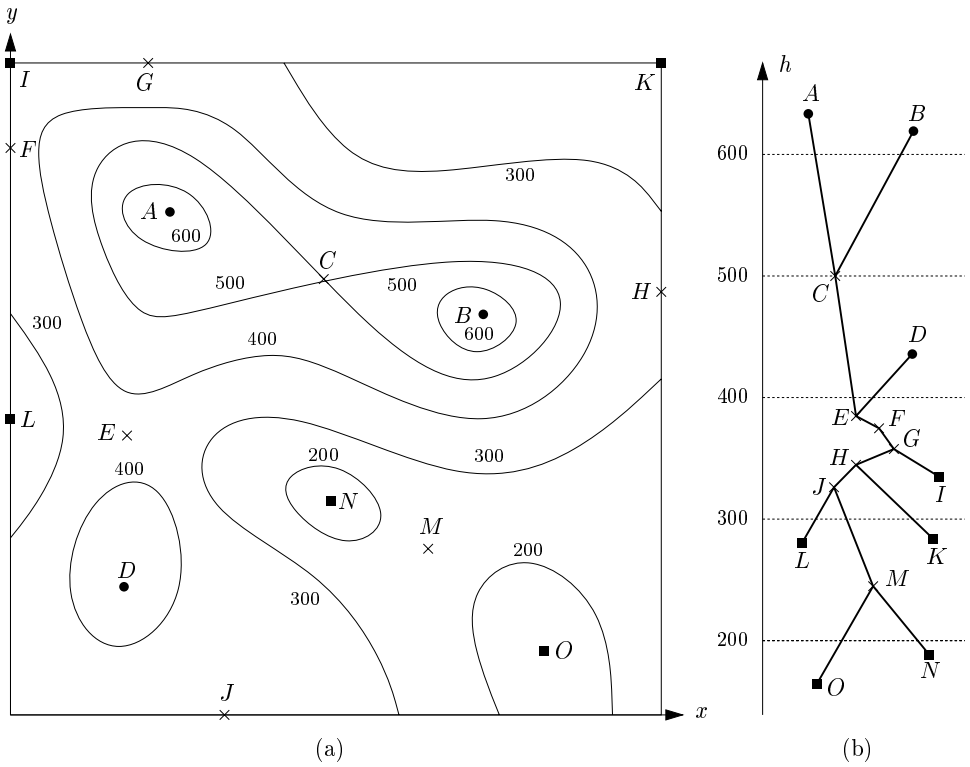

Figure 1: A contour map and the corresponding contour tree. Minima and maxima are indicated by squares and circles, and crosses denote saddle points.

dimensional meshes with the Betti numbers in additional $O(N \log N)$ time. The contour tree already gives β_0. β_2 can be obtained by checking the boundary — a closed surface encloses a void, an open one does not. β_1 can than be obtained by computing the Euler characteristic χ and than solving the formula $\chi = \sum_{i=0}^{d-1} (-1)^i \beta_i$.

2 Definitions

The Input. The input for the algorithm consists of a d-dimensional simplicial complex S which forms a bounded volume $D \subset \mathbb{R}^d$. The scalar field is represented by a function $f : D \to \mathbb{R}$ which assigns a scalar value to every vertex in S and interpolates the rest of D linearly.

N denotes the number of d-dimensional simplices, so the size of the input is $O(N)$. The number of vertices in S is denoted by n. In d dimensions $N = O\left(n^{\lceil d/2 \rceil}\right)$ but for "nicely shaped" D, a small triangulation can be found such that $N = O(n)$.

It is assumed for simplicity that the values of f are distinct over all vertices in S. The same offect can easily be achieved e.g. by ordering points lexicographically after their coordinates if they have the same function value.

Critical Points. In this paragraph the term *neighborhood* $N(v)$ of a vertex v denotes the subgraph of S induced by the vertices in S adjacent to v with v being excluded. Let $N_+(v)$ and $N_-(v)$ be the subgraphs of $N(v)$ induced by the vertices w with $f(w) > f(v)$,

$f(w) < f(v)$ respectively. $C_+(v), C_-(v)$ denote the number of connected components of $N_+(v), N_-(v)$.

A *local maximum* obviously only has lower neighbors, therefore $C_+ = 0$, while a *local minimum* only has higher neighbors so $C_- = 0$. A *regular point* v is a point with $C_+(v) = C_-(v) = 1$. All other points are *saddle points*. The saddle points together with the local extrema are the *critical points*. Topological changes, including changes in the number of components, only occur at critical points.

Using these definitions, it is possible to determine the type for every point in S locally by scanning its neighborhood. This takes $O(N)$ time.

3 The Contour Tree

Join Tree and Split Tree. The upper level set, the lower level set, and the (equality) level set of f at value h are defined as

$f_>(h) := \{ x \in D \mid f(x) > h \}$,
$f_<(h) := \{ x \in D \mid f(x) < h \}$,
$f_=(h) := \{ x \in D \mid f(x) = h \}$.

A *contour* is a connected component of a level set $f_=(h)$. Sweeping through the data by decreasing h, the set $f_>$ grows while $f_<$ shrinks continuously, and $f_=$ is their common boundary. The *join tree* represents the evolution of the components of the set $f_>(h)$ as h varies. The *split tree* is defined similarly for sweeping by increasing h.

Looking at the components of $f_>(h)$ as h decreases the following events can happen.

(a) A new component may appear, starting out at a local maximum.

(b) Several components may merge into a single one at a saddle point.

(c) There may be topological changes which do not change the number of components

Also, events of types (b) and (c) may occur together.

Join Tree Construction. Assume all saddle points are known and they are sorted by their function value. We scan all saddle points v in decreasing order of $f(v)$. For each v, we select a neighbor w in each of the $C_+(v)$ components of $N_+(v)$. We process each w by starting a monotone increasing path at w, continuing until we get stuck in a local maximum or we hit a previously visited vertex.

To maintain the connected components a UNION-FIND data structure is used with the set of saddle points as the ground set. If a monotone path hits a vertex z which was already visited, we FIND its component. If z is not already in the same component as v, we add an edge from v to the lowest vertex in the component of z to the join tree and we perform the UNION of the components containing v and z.

The case when a monotone path ends in a local maximum r is easy: We simply add an edge from v to a new vertex representing r to the join tree.

It may happen that a saddle point receives only one outgoing edge in the join tree because it just reflects a change in topology and not in connectivity (like a change from a torus to a sphere). We leave these vertices of degree two in the tree during the construction, and can eliminate them in a final purification step after combining the join and split tree to the contour tree.

Theorem 1 *Creating the join tree takes time for saddle points identification, for sorting and for the monotone paths. This is in total* $O(N + t \log t + m + s\alpha(s,t))$ *time for* t *saddle points,* m *denoting the number of edges in* S *and* $s = \sum^t C_+(v)$ *is the sum over all higher neighboring components of all saddles. This simplifies to* $O(N + t \log t)$.

The theorem holds for the split tree respectively.

Combining Two Trees. The Algorithm to combine the join tree with the split tree is explicitly shown in [1] and it takes $O(t)$ time for t nodes in the final contour tree.

4 Computing Betti Numbers

Contour Tree with All Critical Points. Usually the contour tree only gives information about connectivity which is sufficient for drawing connected

components. In three and higher dimensions saddle points exist which do not change the number of connected components. Following the algorithm in section 3 these saddle points are nodes in the contour tree with degree two and in this section the tree containing all critical points is refered to as *ECT* — extended contour tree.

Algorithmic Idea. The following only holds for up to three dimensions. The ECT provides all points which cause changes in topology and also the information on which connected component the change occurs. This is sufficient for calculating the Betti numbers for every possible level set.

Assume the critical points v_i ordered by decreasing function value. The Betti numbers for every interval $(f(v_i), f(v_{i+1}))$ are stored in a tree structure with the following simple loop.

The nodes in the ECT are processed from top to bottom. For every node the type is determined and the Betti numbers for the next interval are changed according to that type.

For every node in the ECT the type is known after the construction of the ECT. Possible types are local maximum or local minimum, a saddle uniting two components in the join tree (*join*) or uniting a component with itself (*pseudo-join*). The same types of saddle points can occur in the split tree denoted by *split* and *pseudo-split*.

Betti Numbers for Volumes. The object of interest may be the volumetric object $\{x \in D \mid f(x) \geq w\}$ instead of the surface $f^{-1}(w) = \{x \in D \mid f(x) = w\}$. In this case, parts once connected always stay connected.

For volumetric objects the edges do not always correspond to the number of components. The edges from a split are complementary components also known as voids.

In a pre-processing step, every leaf in the ECT corresponding to a boundary vertex in S has to be removed if the neighbor has a higher function value. This has to be done iteratively, as new such leafs may appear during the process. This removes the topological changes on the "outside" of the object.

The following changes occur at critical points sweeping from higher to lower function values.

Local maximum v_i: A new component starts at $f(v_i)$ and at $f(v_i) - \varepsilon$ it becomes a solid ball. This implies to increment β_0 and β_1, β_2 are not changed.

Join: Several components are connected. β_0 decreases by the number of edges to nodes with higher function value. β_1, β_2 are not changed.

Pseudo-join: A component is connected to itself (maybe several times) which creates one or more "handles" and thereby increases the number of tunnels. β_1 is increased by the number of how many times the component is connected to itself.

Pseudo-split: A tunnel or several ones through the object close. This is the analogous case to a pseudo-join, so β_1 is decreased in the same fashion.

Split: A ball in the complement is split into several ones. The resulting number of voids is one for each neighboring node with lower function value.

Local minimum v: A void becomes very small for $f(v)+\varepsilon$, is only a point at $f(v)$ and vanishes at $f(v)-\varepsilon$. Therefore β_2 decreases by one.

Betti Numbers for Surfaces. Every closed surface contains a void, therefore a component usually increases the number of components and the number of voids. A surface with boundary is open and therefore does not contain a void. A surface can only be open if it touches the boundary of D. This is easy to check if the contour tree does not only contain all critical points but also special boundary vertices of S. This increases the runtime of the contour tree algorithm.

References

[1] Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. In *11th ACM/SIAM Symposium on Discrete Algorithms*, pages 918–926, 2000.

[2] Valerio Pascucci. On the topology of the level sets of a scalar field. In *Lawrence Livermore National Laboratory technical report UCRL-JC-142262*, February 2001.