Simple and Optimal Output-Sensitive Construction of Contour Trees Using Monotone Paths

Yi-Jen Chiang [a,1], Tobias Lenz [b,2], Xiang Lu [a,3], Günter Rote [b,2]

[a] *Department of Computer and Information Science, Polytechnic University, Brooklyn, NY 11201, USA.*

[b] *Institut für Informatik, Freie Universität Berlin, Takustraße 9, D-14195 Berlin, Germany*

Abstract

Contour trees are used when high-dimensional data are preprocessed for efficient extraction of isocontours for the purpose of visualization. So far, efficient algorithms for contour trees are based on processing the data in sorted order. We present a new algorithm that avoids sorting of the whole dataset, but sorts only a subset of so-called *component-critical points*. They form only a small fraction of the vertices in the dataset, for typical data that arise in practice. The algorithm is simple, achieves the *optimal output-sensitive* bound in running time, and works in any dimension. Our experiments show that the algorithm compares favorably with the previous best algorithm.

Key words: Computational topology, level sets, piecewise linear Morse theory.

Email addresses: `yjc@poly.edu` (Yi-Jen Chiang), `tlenz@inf.fu-berlin.de` (Tobias Lenz), `xlu@photon.poly.edu` (Xiang Lu), `rote@inf.fu-berlin.de` (Günter Rote).

[1] Research supported in part by NSF CAREER Grant CCR-0093373, NSF Grant ACI-0118915, and NSF ITR Grant CCR-0081964.

[2] Partially supported by the IST Program of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2000-26473 (ECG—Effective Computational Geometry for Curves and Surfaces).

[3] Research supported by NSF Grant CCR-0093373.

1 Introduction

1.1 Visualization by Isosurfaces

In many scientific fields huge amounts of spatial data are produced, either by measurements or by computer simulations. Examples are electromagnetic waves in nuclear spin tomography, heat distribution in fuel cells, or molecular structures in X-ray crystallography. The field of Scientific Visualization is concerned with methods that help the user to extract meaningful information from the data in a visual and intuitive way.

One way to visualize data is by displaying *level sets* or *isocontours*, which are also called *isosurfaces* for volumetric data and *isolines* for planar data [20,21]. Specifically, for a volumetric dataset, which consists of tuples $(v, f(v))$ where v is a 3D sample point representing a vertex of the input mesh and f is a scalar function defined over 3D points, the *level set* or *isosurface* at *isovalue* h is a set of points (a surface) in the volume whose scalar function value is h; similarly for planar datasets and isolines. Figure 1a shows isolines of a bivariate function $f(x, y)$ and Figure 4 shows some examples of isosurfaces. For volumetric data, it may not be possible to view many isosurfaces at different isovalues simultaneously because they may occlude each other. Therefore it is

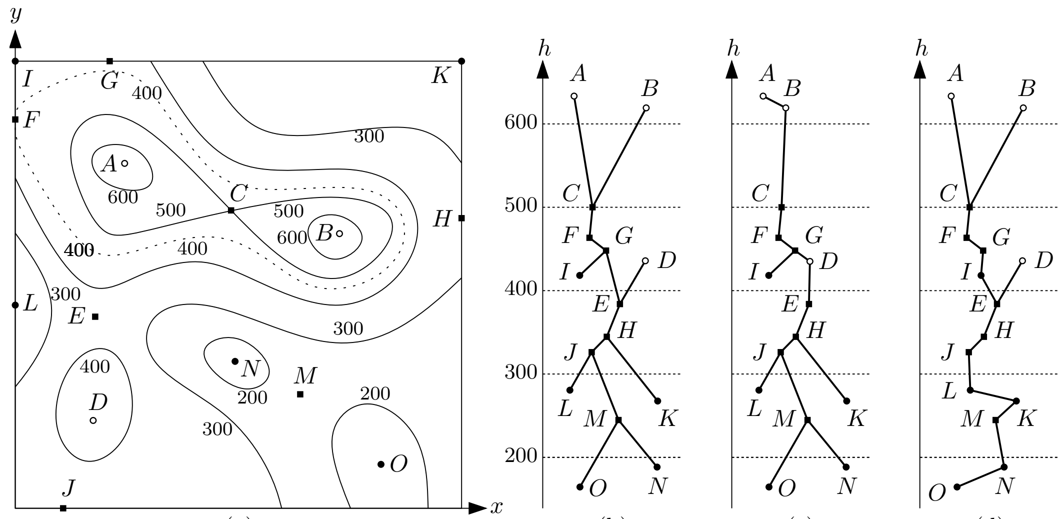


Fig. 1. (a) a contour map of isolines, (b) the corresponding contour topology tree, (c) the join tree, and (d) the split tree. Minima and maxima are indicated by full and empty circles, and squares denote saddle points. The isolines in (a) are labeled with the isovalues, and these isovalues are indicated in the trees of (b), (c), and (d). The critical point F changes only the topology of a contour and not the number of contours; the contour tree is the tree in (b) with the degree-two vertex F removed between C and G.

important for the user to select the isovalues interactively in order to get an intuitive understanding of the data. An efficient interactive system must be able to display isocontours quickly.

Isosurface extraction is one of the most effective and powerful techniques for the investigation of volumetric datasets. It has been used extensively in scientific visualization applications such as biology, medicine, chemistry, computational fluid dynamics, and so on [16]. It has also found extensive applications in computer graphics such as simplification [19] and implicit modeling [33].

1.2 The Contour Tree and the Variants

1.2.0.1 The Contour Tree. The *contour tree* is a fundamental data structure that represents the relations between *contours* (i.e., connected components of the level sets) embedded in a dataset. Connected components that merge together or split as one continuously changes the isovalue are represented as edges that join or branch apart at a node of the tree; see Figure 1b.

This structure was used by Bajaj and van Kreveld *et al.* [3, 41] to speed up isosurface extraction in the *seed-cell propagation* paradigm: given an isovalue, one generates the isosurface by exploring the neighboring cells in the volume starting from the *seed cells*. To guarantee that no connected component of the isosurface would be missed, one has to find a seed cell at the given isovalue for every edge of the contour tree which is intersected at this isovalue.

1.2.0.2 The Contour Topology Tree. As one continuously changes the isovalue, there may be other topological changes of the contours besides splitting and joining: a contour might touch itself and form a ring, thus transforming itself into a surface of different genus, or a contour might touch the boundary and develop a hole. If we subdivide the edges of the contour tree by additional nodes which represent these topology changes, we obtain the *contour topology tree*, which records the complete information about topology changes of the level sets. This tree has been introduced by Pascucci [30, 32] under the name *augmented contour tree* (ACT). The same term has, however, been used in [8] to denote the subdivision of the contour tree by *all* vertices of the input mesh. Therefore, we have chosen the more specific term *contour topology tree*.

In two and three dimensions, we will give a complete classification of all *critical points* at which these topological changes occur, and this will allow us to build the contour topology tree.

1.2.0.3 Surface Networks and the Morse-Smale Complex. In the area of topological surface modeling by *surface networks*, one constructs the *ridges* and *ravines* by following the lines of steepest ascent and steepest descent from saddle points. For a smooth surface (and simple saddles), these curves generically extend all the way to the maxima and minima without meeting, and they dissect the surface into quadrilateral faces which form the *Morse-Smale* complex. Like the contour tree, the Morse-Smale complex is a structure which can be used as a tool to analyze the geometry and topology of a surface. This concept was recently extended to piecewise linear functions on a triangulated surface [15]. There is a close similarity with our algorithm, since both our algorithm and the algorithm of [15] follow monotone paths. However, we have not explored the relation between contour trees and Morse-Smale-complexes more deeply. Moreover, the connection does not extend to higher dimensions, as the Morse-Smale complex of a three-dimensional manifold is formed by *surface patches* which partition the given domain into volume pieces, whereas for the contour trees, we are interested in one-dimensional structures.

1.2.0.4 Further Applications. The succinct encoding of the isosurface topologies in the contour tree also leads to other important applications. Bajaj *et al.* [2] proposed the display of the contour tree to provide the user with insights into the topological structures of the isosurfaces embedded in the volume data. This approach was further explored and extended by Carr and Snoeyink [7]. Pascucci and Cole-McLaughlin [32] gave an elegant algorithm for computing and associating the Betti numbers $(\beta_0, \beta_1, \beta_2)$ with the contour topology tree so that the isosurface topologies, including the number of connected components and the genus number, can be completely determined and displayed; see also [30, 31]. Recently, Chiang and Lu [9] developed a volume simplification technique that preserves all isosurface topologies by making use of the contour topology tree.

1.3 Previous Work

We assume that the scalar function f is defined on an input mesh with n vertices and N cells, and m edges. For *structured meshes*, the data points form a regular grid and the underlying cell topology of the mesh is a cube or a box, while for *unstructured meshes* the vertices are irregularly sampled (with higher resolutions at places of more features) and the underlying cell topology is a simplex (e.g., a tetrahedron in the 3D case).

The first efficient algorithm for computing the contour tree in 2D was given by de Berg and van Kreveld [14], with running time $O(N \log N)$. The algorithm makes a single pass over the data. It sweeps from the highest to lowest function values through the sorted data and keeps track of the level set components which are cut by the sweep plane in its current position.

Independent of this work, Takahashi *et al.* [36] constructed discrete analogs of surface networks formed by *ridges* and *ravines* for piecewise linear functions of two variables, starting at all saddle points and simply following a path from each vertex to the highest or lowest neighbor, respectively. They described an algorithm to convert the resulting surface network into the contour tree, but no run-time analysis was given. (According to our own analysis, it takes $O(N)$ time for reading the data, $O(nt')$ time for finding the paths and $O((t')^2)$ time for constructing the tree, where t' is the number of critical points.) The approach has been extended to three dimensions in a technical report [37], using the term "Volume Skeleton Tree" for the contour topology tree.

Following de Berg and van Kreveld [14], Bajaj and van Kreveld *et al.* [3, 41] developed algorithms for computing the contour tree in 2D with the same running time $O(N \log N)$ (but simpler) in the plane and with $O(N^2)$ time in higher dimensions. They also showed how to compute a small seed-cell set for fast isosurface extraction using the contour tree. Tarasov and Vyalyi [38] improved the complexity of computing the contour tree in 3D to $O(N \log N)$ time.

Carr *et al.* [8] simplified and extended the method of Tarasov and Vyalyi [38] so that the contour tree can be computed in any dimension in $O(N + n \log n)$ time. They used two separate sweeping processes resulting in a *join tree* and a *split tree*, which can then be merged to obtain the contour tree. We review this simple and elegant algorithm in Section 4.2. It is the basis for our new algorithm.

Cox *et al.* [13] defined the *criticality tree* which is comparable with "one half" of the contour tree, namely the *join tree*, augmented with the critical points.

Pascucci and Cole-McLaughlin [32] extended the construction of contour trees from linear interpolation to interpolants of higher degrees. Independent of our work and almost around the same time, Carr and Snoeyink [7] proposed the use of *path seeds* in the seed-cell propagation paradigm [3,41] for fast isosurface extraction mentioned above. A more extensive overview about the history and literature of contour trees is given in the paper [7].

1.3.0.5 Output-Sensitivity. The first *output-sensitive* algorithm for computing the contour tree was given by Pascucci and Cole-McLaughlin [32], with running time $O(n + t' \log n)$, for 3-dimensional *structured meshes* with t' critical points. Note that for structured meshes, $N = O(n)$ and thus N does not appear in the bound, and that t' is an upper bound on the size of the contour tree. As mentioned above, they also gave a very nice method to associate the Betti numbers to the contour tree to enable the computation of isosurface topologies.

This contour-tree algorithm is based on the divide-and-conquer paradigm. At each step, the volume is recursively subdivided into two halves of roughly equal number of vertices, with the common boundary (the *separator*) having $O(n^{2/3})$ vertices and edges. This splitting step can be performed trivially on a *structured mesh* in $O(1)$ time—just take a plane cutting through the median of the volume. In each half, the corresponding contour tree is built recursively, and then the two contour trees, one on each side, are merged to form the final contour tree. In the merge step, the two contour trees from the two sub-volumes can interact with each other only at the $O(n^{2/3})$ vertices on the separator; for the rest of the two contour trees we just need to copy them, which takes $O(t')$ time at each recursive level. Therefore, the merging step takes $O(n^{2/3}\alpha(n)+t')$ time (by merging the separator vertices from both sides into a combined sorted list and then performing the sweeping algorithm of Carr *et al.* [8] with UNION-FIND operations [12], plus copying the unchanged portions of the two contour trees). Here, $\alpha(n)$ is the extremely slowly growing inverse of the Ackermann function. Since the merge step (as well as the splitting step) takes less than linear time in n, the overall complexity is $O(n + t' \log n)$.

In [32] the authors also mentioned that their method might be extended to *unstructured meshes*, by pointing out the $O(n)$-time algorithms [26, 27] for finding a separator with $O(n^{2/3})$ vertices in unstructured meshes. However, even with these separator algorithms, the output-sensitive bound no longer holds: The volume splitting step, originally taking $O(1)$ time in each recursion for structured meshes, now takes $O(n)$ time —the overall complexity for all splitting steps is $O(n \log n)$ already, and thus it does not pay off to use the more complicated separator algorithms [26,27], compared to the simple sorting step used in the algorithm of Carr *et al.* [8].

1.4 Our Algorithm

We will follow the idea of Carr *et al.* [8] of constructing the join tree and the split tree separately. Our algorithm differs in the way how these trees are constructed. The key idea of our algorithm is to avoid sorting all vertices. Instead, we identify all critical points first, sort them, and then connect them appropriately. In order to determine the connectivity of the components which meet (locally) at a vertex v, we start a monotone path from v into every component until we hit a previously visited vertex. It is very easy to follow a monotone path, by just searching through the neighbors.

The main advantage is that only the *component-critical points* have to be sorted. These points must be found in an initial (unordered) scan of the data. A component-critical point can be identified by looking at its neighbors. The running time of our algorithm is $O(N+t\log t)$ if there are t component-critical points. (The component-critical points form a subset of all critical points and thus $t \leq t'$ if there are t' critical points.) The algorithm works for structured as well as for unstructured meshes, in any dimension. It is *output-sensitive*, and is simple and easy to implement. In addition, as a by-product, the monotone paths implicitly give a reasonable seed set, to be used in the seed propagation paradigm [3, 41] for fast isosurface extraction. In two and three dimensions, an easy extension can compute the contour topology tree in $O(N + t' \log t')$ time if there are t' critical points.

It can be shown that there is a lower bound of $\Omega(N+t\log t)$ on the complexity of computing the contour tree by the lower bound construction of Bajaj *et al.* [3]; see Section 5.1 for the details. Thus, our algorithm is *optimal* in the worst case.

We have implemented our algorithm and performed experiments on datasets from real-world scientific visualization applications. The experiments show that typically t is less than 5% of the overall number n of the input vertices, and that our algorithm compares favorably with the previous best algorithm [8]. We refer to Section 6 for the details.

Our method has additional advantages for huge datasets that do not fit into main memory. This is discussed in the concluding section (Section 7).

Our algorithm bears some resemblance to the approach of Takahashi et al. [36, 37], as both algorithms follow monotone paths. There are some differences, however. In [36], one always follows the "steepest" path (to the highest or lowest neighboring vertex) with the objective to approximate critical flow lines, i.e., ridges and ravines, and one follows these lines all the way to the maximum or minimum. In our algorithm, we can follow an arbitrary monotone path, and we stop when we reach a visited vertex. (In our computational experiments, we also follow steepest paths, but the objective is to (heuristically) keep paths short.) Also, by combining our algorithm with the techniques of [8], we are able to obtain a simple algorithm with a provably good running time.

1.4.0.6 Overview. Section 2 states the basic definitions and assumptions. Section 3 is devoted to the definition and characterization of critical points. We describe our algorithm in Section 4, present the analysis of its running time in Section 5, and report the experimental results in Section 6. Section 7 gives comparisons with previous algorithms and lists some extensions and further questions.

This work was started concurrently and independently by Yi-Jen Chiang and Xiang Lu [10], and as the Master's thesis of Tobias Lenz [25] under the supervision of Günter Rote. The results were presented at the 19th European Workshop on Computational Geometry in Bonn, March 24–26, 2003.

2 Definitions

We assume that the scalar function f is specified as a piecewise linear interpolated function on some n data points V. We are given a triangulation T of a domain $M \subset \mathbb{R}^d$ with vertex set V, and the values $f(v) \in \mathbb{R}$ for $v \in V$. The triangulation T can be given explicitly as a list of N d-simplices (an *unstructured mesh*), or, when the data points V form a grid (a *structured mesh*), the triangulation can be implicitly given by some fixed triangulation of the grid following a regular pattern. For an unstructured mesh we assume that each simplex is linked to its $d+1$ neighboring simplices. For a structured mesh, we will have $N = O(n)$. In the worst case, $N = O(n^{\lceil d/2 \rceil})$, although we do not think that triangulations with super-linear N will be very useful in this context.

We have to assume that M is a simply-connected d-dimensional manifold with boundary. For example, a convex domain M will do. The graph of the triangulation (consisting only of the vertices and edges) is denoted by $G = (V, E)$.

2.0.0.7 Technical Assumption. We assume that all n values $f(v_i)$, $v_i \in V$, are distinct. If we would allow values at adjacent vertices to be equal, the notion of critical points has to be adapted, and this causes technical difficulties. Given the intended purpose of contour trees (visualization of measured data), it is appropriate to resolve such cases by perturbation, see Section 4.3. (However, Cox *et al.* [13] have managed to extend the notion of critical points and saddles also to the case where the function is constant on a whole d-dimensional cell.)

On the other hand, we could allow non-adjacent vertices to have equal values without difficulty. In this case, our assumption is merely a convenience. These are the same assumptions used in Carr *et al.* [8] or for *piecewise linear Morse functions* in Banchoff [4, 5]. Under these assumptions, the critical points only occur at the vertices $v \in V$ [4–6, 23, 24].

2.0.0.8 Level Sets and Contours. The upper level set, the lower level set, and the (equality) level set of f at *height* h are defined as

$$
\begin{aligned}
M_{>h} &:= \{\, x \in M \mid f(x) > h \,\},\\
M_{<h} &:= \{\, x \in M \mid f(x) < h \,\},\\
M_{=h} &:= \{\, x \in M \mid f(x) = h \,\}.
\end{aligned}
$$

A level set at a certain *height* or *isovalue* h is called *isoline* in a 2D mesh and *isosurface* in a 3D mesh. A *contour* is a connected component of a level set $M_{=h}$. As we sweep through the data by increasing h, the set $M_{<h}$ grows while $M_{>h}$ shrinks continuously, and $M_{=h}$ is their common boundary. The *join tree* represents the evolution of the components of the set $M_{<h}$ as h varies. The split tree is defined similarly for $M_{>h}$.[4] See Figures 1c and 1d.

The contour tree is obtained by contracting each contour to a point while preserving the adjacency information between contours. In topology, this structure is also known as a *Reeb graph*. In general, it can be a graph of arbitrary structure, but if the underlying domain is simply connected, this graph is a tree [3, 41]. Each leaf node represents a local minimum or maximum of f, at which a contour appears or disappears, and each interior node represents the joining and/or splitting of two or more contours at a *component-critical point* (to be defined in Section 3). Reeb graphs for surfaces which are not simply connected were recently considered in [11], where an algorithm for constructing the Reeb graph in $O(n \log n)$ time was presented.

3 Critical Points

In classical (smooth) Morse theory [28, 35], a critical point is a point where the gradient vanishes. The basic theorems of Morse theory assert (under some additional assumptions) that these points are precisely the points where the topology of the level sets change. For piecewise linear functions, we therefore mean by a *critical point* a point where the topology of the level set changes as the height passes through this point. A precise definition is given below in Theorem 1. A point which is not critical is called a *regular point* or *ordinary point*.

In this section, we show how to identify critical points locally, for 2- and 3-dimensional manifolds. It turns out that it is only necessary to count connected components of lower and higher neighbors of a vertex. In dimension four and

[4] The names join tree and split tree refer to the convention of sweeping from lower to higher values. The sweeping direction is not consistent in the literature, therefore the terms *join* and *split* are used inversely in other papers.

higher, it appears more difficult to test whether a point is regular or critical. We discuss this issue in the conclusions, Section 7.

The *link graph* $N(v)$ of a vertex v in a triangulated manifold M is the graph obtained by taking all vertices and edges of the simplices containing v, and removing v (and the edges incident on v). In topological terms, this is the skeleton of the link of v.[5] Let $N_+(v)$ and $N_-(v)$ be the subgraphs induced by the vertices w with $f(w) > f(v)$ and $f(w) < f(v)$ respectively, and $C_+(v)$ and $C_-(v)$ respectively denote the numbers of connected components in $N_+(v)$ and in $N_-(v)$. We omit the reference to v when it is clear from the context.

Definition 1 *A* maximum *is a point with* $C_+ = 0$ (*and hence* $C_- = 1$)*. A* minimum *is a point with* $C_- = 0$ (*and hence* $C_+ = 1$)*. A* join candidate *is a point with* $C_- > 1$ (*and hence* $C_+ \geq 1$)*, and a* split candidate *is a point with* $C_+ > 1$ (*and hence* $C_- \geq 1$)*.*

Note that the notions of join candidate and split candidate are not exclusive: a vertex with $C_- > 1$ and $C_+ > 1$ is simultaneously a join candidate and split candidate.

In two dimensions, these are all critical vertices. In a trivariate function defined over a bounded domain, we have to introduce additional types of critical points:

Definition 2 *Let v be a point with $C_- = C_+ = 1$. Then v is a* boundary minimum (boundary maximum) *if v is on the boundary of M and is a minimum* (*maximum*) *among the neighbors of v which lie on the boundary.*

Note that C_- and C_+ are taken with respect to the whole neighborhood (as usual), not just restricted to the boundary. It is also important to note that the above definition explicitly excludes points which fall under Definition 1; in particular, minimum/maximum and boundary minimum/maximum are *mutually exclusive.*

Theorem 1 (Critical points in three dimensions) *Let f be a piecewise linear function defined on a simplicial decomposition of a three-dimensional manifold M, and let v be a vertex of this decomposition. Suppose that v is the only vertex with value $f(v)$. Then precisely one of the following two alternatives holds:*

---

[5] The link graph is *not* the neighborhood in the graph-theoretic sense, i. e., the subgraph of the skeleton G induced by the neighbors of v. There might be an edge between two neighbors of v which does not belong to a common simplex with v, in the triangulation. However, the subsequent Lemma 5 would remain true even if we used the graph-theoretic neighborhood instead of the link graph in the definition of component-critical points.

(1) v is a maximum, a minimum, a split candidate, a join candidate, a boundary maximum or a boundary minimum.

(2) There is an $\varepsilon > 0$ such that all level sets $M_{=h}$ are homeomorphic, for $f(v) - \varepsilon \leq h \leq f(v) + \varepsilon$.

PROOF. See Appendix A. □

If the second alternative holds, we call v a *regular point*, otherwise v is a *critical point*, and $h(v)$ is called a *critical value*. Thus we propose the following definition of a regular point in three as well as in two dimensions.

Definition 3 *Let f be a piecewise linear function on a simplicial decomposition of a two- or three-dimensional manifold M, and let v be a vertex of this decomposition with the unique value $f(v)$. Then v is called* regular point *if and only if there is an $\varepsilon > 0$ such that all level sets $M_{=h}$ are homeomorphic, for $f(v) - \varepsilon \leq h \leq f(v) + \varepsilon$.*

For a regular point, the homeomorphism is in fact an isotopy:

Theorem 2 *Under the assumptions of Theorem 1, let $[a, b]$ be an interval which contains no critical value. Then there is an isotopy between all level sets in this range, i. e., a continuous map*

$$g\colon M_{=b} \times [a, b] \to M,$$

where $g(\cdot, h)$ is a piecewise linear homeomorphism between $M_{=b}$ and $M_{=h}$, for every $h \in [a, b]$.

PROOF. See Appendix A. □

Note that g is not necessarily a piecewise linear function of both variables. The condition is also necessary: If $[a, b]$ contains a critical value, the topology changes at this point, and there can be no homotopy.

It seems that such basic statements about piecewise linear functions ought to be known, but we could not find them explicitly in the literature. Therefore, we provide proofs in the appendix.

Table 1 lists the types of critical points. For illustration, there is a typical example of a smooth function for which the origin is a critical point of the respective type. The first four types in their pure form are the generic criticalities of smooth Morse functions. The critical points which are not minima or maxima are the *saddle points*. Note that monkey saddles and saddles of

Table 1
Critical points for a function of three variables.

critical point	C_-	C_+	smooth example	Morse index
minimum	0	1	$x^2 + y^2 + z^2$	0
join candidate	> 1		$x^2 + y^2 - z^2$	1
split candidate		> 1	$x^2 - y^2 - z^2$	2
maximum	1	0	$-x^2 - y^2 - z^2$	3
join and split candidate	> 1	> 1	$z(z^2 - x^2 - y^2)$	
boundary maximum	1	1	$-x^2 - y^2 + z$ $(z \geq 0)$	
boundary minimum	1	1	$x^2 + y^2 - z$ $(z \geq 0)$	

even higher multiplicity are not exceptional for piecewise linear functions, in contrast to the smooth setting of Morse theory.

For the contour tree, we are only interested in the components of level sets, not in their complete topology. So we need to consider only a subset of the critical points: the *component-critical points.*

Definition 4 *A* component-critical point *is a join candidate, a split candidate, a maximum, or a minimum.*

Note that a *boundary minimum/maximum* is *not* a component-critical point. In other words, a vertex which is *not component-critical* is characterized by $C_- = C_+ = 1$. We summarize the criticality classification of a vertex v in a bounded three-dimensional triangulated manifold M in the following procedure.

Procedure *Criticality Classification in 3D.*

(1) If the condition $C_- = C_+ = 1$ does not hold, then v is *component-critical.*
(2) Else ($C_- = C_+ = 1$), we distinguish two cases:
  (a) If v is on the boundary of M and is minimum (maximum) among all its neighbors that lie on the boundary of M, then v is *boundary minimum* (*boundary maximum*). (In this case, v is *critical* but *not component-critical.*)
  (b) Otherwise, v is *regular.*

For the contour tree, Step 1 above suffices to identify all component-critical points, which are all we need to consider: The maxima and minima become leaves of the contour tree, and the join candidates and split candidates are the vertices which may generate join and split vertices in the contour tree. To identify component-critical points in higher dimensions, we show that the following statement holds *in all dimensions.*

Lemma 5 *If h passes a vertex v of a d-dimensional triangulated manifold which is not component-critical, the number of connected components of $M_{>h}$, of $M_{=h}$, and of $M_{<h}$ is unchanged.*

PROOF. It is easy to see that, when restricted to a neighborhood $B_\varepsilon(v)$ of v, $M_{<h}$ has a single component for all values h in some neighborhood of $f(v)$, and the same holds for $M_{>h}$. (A formal argument can be given with the help of Lemma 7 below.) By the d-dimensional (polyhedral) analog of the Jordan curve theorem applied to $B_\varepsilon(v)$, a level set $M_{=h}$ consisting of $k > 1$ components inside $B_\varepsilon(v)$ would partition $B_\varepsilon(v)$ into at least $k+1$ pieces where f is uniformly larger or smaller than h. It follows that $M_{=h}$ must also consist of a single component.

It is clear that the connectivity of the components of $M_{>h}$, $M_{=h}$, and $M_{<h}$ does not change outside $B_\varepsilon(v)$ or when h does not pass a vertex. $\square$

For completeness, we also discuss the situation for two-dimensional manifolds, which is easier. In two dimensions, every join candidate is automatically a split candidate and vice versa, except at the boundary. Each join and split candidate will always be a join or split vertex (or both) in the contour tree when M is a subset of the plane. Candidates at the boundary do not necessarily become vertices of the contour tree. Thus we have the critical points in Table 2. The regular vertices on a two-dimensional manifold are characterized by the condition $C_- = C_+ = 1$.

Theorem 3 (Critical points in two dimensions) *Let f be a piecewise linear function defined on a triangulation of a two-dimensional manifold M, and let v be a vertex of this decomposition. Suppose that v is the only vertex with value $f(v)$. Then v is a regular point if and only if $C_- = C_+ = 1$.* $\square$

The above characterization allows us to find the component-critical points or the critical points in linear time by scanning through the triangulation. For each vertex v, we have to scan the link graph $N(v)$ and split it into connected

Table 2
Critical points for a function of two variables.

critical point	C_-	C_+	smooth example	Morse index
minimum	0	1	$x^2 + y^2$	0
join and split candidate	> 1	> 1	$x^2 - y^2$	1
maximum	1	0	$-x^2 - y^2$	2
split candidate	1	> 1	$x^2 - y \ (y \geq 0)$	
join candidate	> 1	1	$-x^2 + y \ (y \geq 0)$	

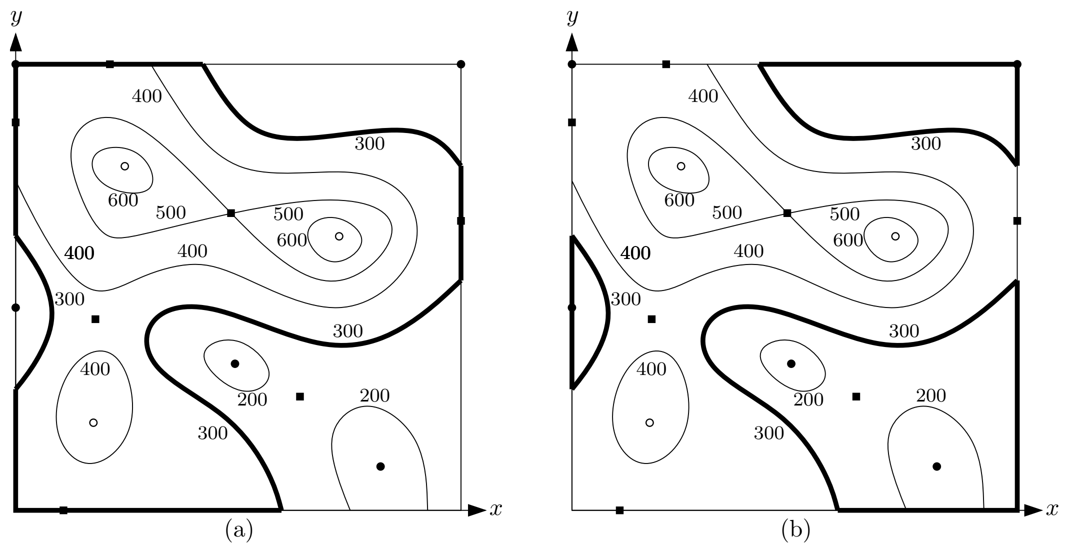


Fig. 2. Two possible treatments of the boundary that correspond to closing the "holes" in the contours meeting the boundary, by setting the function values outside the domain to be (a) $-\infty$ and (b) $+\infty$. We show the effects on the contours at height 300 in the example of Figure 1.

components of the same sign ("+" or "−" depending on whether the function values are larger or smaller than $f(v)$). This can be done in linear time in the size of the graph. Each simplex of the triangulation contributes to only a constant number of edges to all link graphs. (In d dimensions, this number is $d(d-1)(d-2)/2$.) Hence the total size of all link graphs is $O(N)$, if there are N simplices. Thus we have:

Lemma 6 *For a two- or three-dimensional manifold with N simplices, the critical vertices can be identified in $O(N)$ time. In any dimension, the component-critical vertices can be identified in $O(N)$ time.* □

3.1 Treatment of the Boundary

We have assumed that a contour is cut at the boundary of the domain M. Thus, a contour can be a manifold with boundary. Another approach has also been proposed in the literature. One may regard the function f as defined over the whole space $\mathbb{R}^d$ by setting $f(x) = +\infty$ or $f(x) = -\infty$ (or some "very large" or "very small" value) whenever x lies outside M. This corresponds to closing the "holes" in the contours in one of two possible ways; see Figure 2. For example, if f represents a density function, and the density is indeed concentrated well inside M, then it makes sense to set $f(x)$ to zero outside M. The isovalues at which the contour would touch the boundary are so low that this event and the corresponding critical point is not a "critical value" that would be of interest to the user. On the other hand, in case of a turbulence simulation in some portion of space, one might prefer to have the contour surfaces simply cut at the boundary, without "artificially" closing the holes.

If the approach of extending the domain of f and thereby closing the holes at the boundary is taken, all boundary effects on topology of the level sets disappear. In three dimensions, there are no boundary minima and maxima, and component-critical and critical vertices coincide.

We remark that an alternative treatment of the boundary effects in three dimensions has been proposed by Gerstner and Pajarola [17]. At a boundary vertex v, one can essentially "reflect" the neighborhood of v at the boundary to obtain a structure which, together with the original neighborhood, completely surrounds v. Then v can be classified like an interior vertex: v is a regular vertex if and only if $C_+ = C_- = 1$ in the combined neighborhood. The resulting classification gives the same result as our more explicit classification in Theorem 1 (and Procedure *Criticality Classification in 3D*).

It is interesting to see why the "reflection" method [17] gives the same classification result as our Procedure *Criticality Classification in 3D*. By our method, a boundary vertex v is regular if and only if (a) $C_+ = C_- = 1$ (without the "reflection") and (b) v is *not* a boundary minimum or a boundary maximum. If condition (a) does not hold, then v is critical by both methods. Now suppose condition (a) is true. If v has both "+" and "−" boundary neighbors (i.e., v is *not* a boundary minimum/maximum), then the "+" boundary neighbors belong to the single "+" component, and the reflected and the original "+" components are merged into a single component via the "+" boundary neighbors; similarly for the "−" component(s). Therefore we have $C_+ = C_- = 1$ in the combined neighborhood, and v is regular by both methods. On the other hand, if v is a boundary minimum, then its boundary neighbors are all in the "+" component, and thus the original and reflected "−" components cannot be merged, resulting in $C_- = 2$ and $C_+ = 1$ in the combined neighborhood, and v is a critical point by both methods; similarly for a boundary maximum. Therefore the two methods give the same classification result. Our method is more explicit and easier to use; see Chiang and Lu [9] for its use in tetrahedral volume simplification preserving all isosurface topologies.

4 The Monotone Path Algorithm

4.0.0.9 Review: The Sweep Algorithm. We briefly review the algorithm of Carr *et al.* [8] for computing the contour tree. It consists of the following steps.

(1) Sort all n vertices of the mesh by their function values.

(2) Perform a sweep of the n vertices from the smallest function value to the largest function value, and build the *join tree*.
(3) Perform another sweep of the n vertices, now from the largest function value to the smallest function value, and build the *split tree*.
(4) Merge the join tree and split tree together and remove all degree-two nodes in the resulting tree to obtain the contour tree.

Step 2 and Step 3 are completely symmetric, and are essentially the same process performed in the opposite sweeping directions. We will describe the procedure for constructing the join tree after describing our own method for this task. (The authors of [8] make different claims about the running time of their algorithm in various parts of their paper. At the end of Section 4.1 they claim a running time of $O(m+t\alpha(t))$ for t component-critical vertices if there are m edges in the graph $G=(V,E)$. (Recall from Section 2 that G consists only of the vertices and edges of the triangulated mesh.) According to our own analysis, the construction of the join tree from the sorted sequence of n vertices as described in [8] takes $O(m\alpha(m,n))$ time. However, this discrepancy does not influence the overall running time, which is dominated by the $O(n\log n)$ sorting step, when $m=O(n)$.)

Step 4 can be performed in time linear in the total size of the join and split trees:

Theorem 4 ([8, Section 4.2]) *The contour tree can be constructed in linear time from the join tree and the split tree.* □

Including the sorting time in Step 1, and the time for extracting the graph G from the input, the overall time is $O(N+n\log n)$. (The claimed running time of $O(m+n\log n)$ in [8] apparently assumes that the m edges of the graph G are already available in an appropriate data structure. Otherwise it is impossible to solve the problem without looking at the whole input, which is of size $O(N)$. In dimensions higher than three, the number N of simplices can be much larger than the number m of edges.)

A related algorithm (constructing the contour tree from the surface network) is described in [36], but without run-time analysis. By our own analysis, the overall algorithm takes $O(N+nt'+(t')^2)$ time if there are t' critical points.

4.0.0.10 The Monotone Path Algorithm. We give an outline of our new algorithm. The algorithm for the contour topology tree and for the contour tree differ only in the first and the last steps.

(1) For constructing the contour tree, identify the component-critical vertices in the mesh. For constructing the contour topology tree, identify all critical vertices in the mesh.

(2) Sort those vertices by their function values.
(3) Build the join tree: Process those sorted vertices by increasing function values. For each current vertex v_i perform the following:
    (a) Start a *monotone descending path* (defined later) from every component in $N_-(v_i)$ until a vertex which was already visited is found.
    (b) Connect v_i to the appropriate tree component if v_i is not already connected.
(4) Build the split tree: This is symmetric to Step 3. Now process the sorted vertices by decreasing function values.
(5) Merge the join tree and the split tree together. If the contour topology tree is desired, the algorithm stops here.
(6) To get the contour tree, remove the nodes of degree two from the tree.

The identification of the critical or component-critical vertices in Step 1 can be done in $O(N)$ time by Lemma 6. Steps 5 and 6 are identical to Step 4 of the sweep algorithm and take linear time, by Theorem 4. Step 3, which is new, is described in the following section.

4.1 Constructing the Join Tree

We describe the incremental construction of the join tree. We scan all critical or component-critical vertices v in increasing order of $f(v)$. Each v is processed as follows: First we create a new node that represents v in the join tree. If v is a local minimum, processing of v is completed and the new node remains isolated. Otherwise, we select a neighbor w in each of the $C_-(v)$ components of $N_-(v)$. These neighbors are processed sequentially. We process w by starting a *monotone descending path* at w, continuing until we hit a previously visited vertex. A *monotone descending path* is a path on which the function values of the vertices are monotonically decreasing. We will always meet a previously visited vertex, because the only way to get stuck would be in a local minimum. However, this minimum is a component-critical vertex and thus has already been visited previously.

Before describing how to connect the node v in the join tree, we need a little lemma about the connection between connected components of level sets (in the topological sense) and connected components in the graph G. Let $G_{<h}$ be the subgraph of G induced by the vertices v with $f(v) < h$.

Lemma 7 (see [8, Lemma 4.2]) *For any value h, the number of connected components of $M_{<h}$ is equal to the number of connected components of $G_{<h}$ (in the graph-theoretic sense).*

PROOF. A simplex cannot (partially) belong to more than one connected component of $M_{<h}$, and if it belongs to a component of $M_{<h}$, then some vertex of the simplex must belong to $M_{<h}$. On the other hand, all vertices v of a simplex with $f(v) < h$ lie in the same connected component of $G_{<h}$. The lemma follows from these observations. □

Let H denote the graph of all traversed edges in all monotone paths before the current vertex v is processed, including all local minima that were visited. Let H'' denote the same graph *after* the current vertex has been processed, and let H' denote the graph H'' with the vertex v removed. We have $H \subseteq H' \subseteq H''$. The following invariant is maintained throughout the algorithm.

Lemma 8 *Both the graph H and the graph H' has precisely one connected component in every component of $M_{<f(v)}$.*

PROOF. This is proved by induction. The lemma holds for the smallest vertex v because H, H', and $M_{<f(v)}$ are all empty.

Assume that the lemma holds for the graph H for the current vertex v. H' is obtained from H by starting monotone paths from certain neighbors w of v with $f(w) < f(v)$. Each path starts in one component of $M_{<f(v)}$, and since we only walk downwards, the path cannot leave that component. So it will connect to the corresponding component of H. Thus the lemma holds for H', too.

We will now show that the statement of the lemma holds for H'' and $M_{<f(v)+\varepsilon}$, for some small ε.

When we increase h from $f(v)$ to $f(v) + \varepsilon$, all components of $M_{<h}$ in the neighborhood of v become connected. This is reflected in H'' because H'' contains an additional edge from v to a vertex w in each component of $M_{<h}$ in the neighborhood of v. If v is a minimum, a new component appears in $M_{<h}$, and at the same time, H'' contains a new isolated vertex. Thus, in any case the statement of the lemma holds for H'' and $M_{<f(v)+\varepsilon}$.

H'' is the graph H for the next vertex $\hat{v}$ that will be processed. By Lemma 5, the component structure of $M_{<h}$ remains unchanged if we increase the height h further until it reaches $f(\hat{v})$. Thus, the lemma holds for the graph H for the next vertex $\hat{v}$. □

We now describe how we manage the connected components of H and how we build up the join tree. The natural representation for maintaining connected components under insertions is of course a UNION-FIND data structure. Our ground set is the set of critical or component-critical vertices determined in Step 1. We choose as a *representative* of a component of H the highest critical vertex (that was added last) in that component. This critical point is at the same time the highest point in the component of the *join tree* that has been constructed so far, in which all ascending paths (in the join tree) in that component come together.

As we walk along a monotone path from a neighbor w of v, we mark the visited vertices and give them a pointer to v. (The vertex v will become the highest vertex in their component.) The vertices on the path do not become part of the UNION-FIND structure; their pointers are never changed.

Now, if a monotone path hits a vertex z which was already visited, we first follow the pointer from z to its (component-)critical vertex (unless z happens to be a (component-)critical vertex itself). Then we query this (component-) critical vertex in the UNION-FIND structure and FIND its representative r. If it turns out that z is already in the same component as v ($r = v$) we do not have to do anything else. Otherwise, we add an edge from v to r into the join tree, perform the UNION of the components containing v and r, and select v to be the representative of the resulting set. In this way we ensure that the representative of a component is always the highest vertex of that component.

To show correctness, we can note that the algorithm maintains the following invariant:

Lemma 9 *At the time when a vertex v is about to be processed, we have constructed all parts of the join tree with values below $f(v)$. In each component of the partial join tree, there is a highest vertex which is reachable from all other vertices in that component by a monotone ascending path.*

PROOF. We proof the lemma by induction. The base case is trivially true. For the induction step, we see that the edge between v and r is the *correct* edge that must be added to the join tree: This edge represents the single connected component of $M_{<h}$ containing r as h varies from $f(r)$ to $f(v)$, since r is the highest point in the component of the join tree that corresponds to this component. The parts of the join tree below r have already been constructed by induction hypothesis.

After processing v, all such edges (v, r) described above are correctly added to the join tree, and the components of the partial join tree below $f(v)$ that belong to the component of v are all connected to v and have v as their highest vertex in the join tree. Thus, the invariant stated in the lemma is maintained. □

4.1.0.11 Seed Sets. When we add a new edge (v, r) to the join tree, we can also remember the initial portion of the monotone path in the mesh, from

v down to the height of r, as a possible seed set for this tree edge (v, r). This is not necessarily a very good, i.e., small, seed set, but it has the nice property that the successive h-intervals on the path are disjoint, except for the last edge, whose interval may overlap with other intervals. The selection of optimal or approximately optimal seed sets was discussed in [3, 41]. However, it is more in line with our minimalist approach to be satisfied with reasonable seed sets that are given by simple heuristics, instead of spending much time striving for the optimum.

If we associate the sorted list of seed edges with every contour tree edge we can find seed edges for a given isovalue h by first determining the affected edges of the contour tree, and then for each such edge, find a seed edge by binary search. This approach is also used for the "flexible isosurface" of Carr and Snoeyink [7] which allows to select contours of different isovalues in different branches of the contour tree.

Carr and Snoeyink [7] also proposed a method which reduces the amount of storage for storing seeds at the expense of running time when constructing seeds: They store only the first edge and re-generate the path for a single contour tree edge on the fly when a seed at a given height is required. In this way the storage is reduced to $O(t)$. The additional time for seed extraction is often dominated by the actual contour extraction which follows it.

A more flexible trade-off between storage and construction time is offered by the following idea. If we store every, say, 10th edge of a monotone path, then a seed at a particular isovalue can be found after scanning at most 10 successive edges of the monotone path (provided that a deterministic strategy for finding paths is used).

4.2 Constructing the Join Tree by the Sweep Algorithm

The original sweep algorithm of Carr *et al.* [8] for computing the join tree can be described as a variation of the monotone path algorithm where *all* vertices are processed, not only the (component-)critical vertices. In this case, there is no need to walk along monotone paths, as each neighbor w has already been visited and the path can stop there. Also, it is easier to just visit *every* lower neighbor w of the current vertex v instead of finding connected components in $N_-(v)$ and picking one neighbor in each component.

The treatment of the connected components using the UNION-FIND structure is then essentially the same as described above for the monotone path algorithm.

4.3 Precision and Degeneracy

Our algorithm only compares the function values of the mesh vertices and does not perform any calculations with them. Therefore, round-off error in floating-point calculations is not an issue.

A degenerate case for our algorithm is given when two vertices have the same scalar function value. This can be resolved with an easy perturbation scheme. In the implementation we used the following simple symbolic perturbation: If two vertices u and v have the same scalar value $f(v) = f(u)$, then we use the vertex ID's to break the tie. The tie breaking is needed both in the critical-point identification and in the sorting steps of our algorithm.

This symbolic perturbation also ensures that the scalar values at all vertices are distinct, to satisfy the technical assumption of Section 2.0.0.7 and to guarantee that critical points only occur at mesh vertices.

4.4 A Possible Simplification

There is a possible simplification, which, however, does not improve the asymptotic complexity. Therefore we only sketch it briefly.

When generating the join tree, one just has to process the sorted list of join candidates, and when generating the split tree, one just processes the sorted list of split candidates. Maxima and minima are not processed explicitly. When walking along a monotone descending path, it may happen that one gets stuck in a local minimum before hitting a previously visited vertex. Then one must create a new vertex for this minimum in the join tree. In this way all minima are eventually discovered.

In the correctness proof, the statement of Lemma 8 must be modified: The components of H and H' lie in all components of $M_{<h}$ except those that contain only a single (component-)critical point (which must then be a local minimum).

There is a slight twist when the join tree and the split tree are combined into the contour tree: the join tree and the split tree have different sets of vertices. The join tree does not contain any maxima, and the split tree does not contain any minima, and some other vertices may be present in only one tree. However, the algorithm of [8] mentioned in Theorem 4 can be adapted without much difficulty to accommodate this, see [8, end of Section 4.2]. It may even happen that the two trees have completely disjoint sets of vertices, in which case some special treatment is required. The details are described in [25, Chapter 7].

5 Analysis

The first step in the algorithm is to identify the critical or component-critical vertices (see Section 3). This can be done in $O(N)$ time by Lemma 6.

The time for exploring the monotone paths is proportional to the number of edges visited, which is bounded by the total number m of edges of the graph G. But in practice, the number of visited edges is much smaller than m; see Section 6.

If there are t component-critical points $v_1, v_2, \ldots, v_t$, and a total of $s = \sum_{i=1}^{t} C_-(v_i)$ of their neighbors are processed, the algorithm performs at most t UNION and at most s FIND operations to construct the join tree. The dominating step is usually the walking of the paths, because this is repeated for each of the s neighbors, and the paths may be very long. But as mentioned above, in the worst case this is bounded by the total number m of edges of the graph G, which is $O(N)$ and thus dominated by the initialization time. All other operations are linear in t and s. Including the time for sorting the t component-critical points, the time is

$$O(N + t\log t + s\alpha(s,t)) = O(N + t\log t),$$

where $\alpha(s,t)$ is the extremely slowly growing inverse of the Ackermann function. We use the version from Tarjan's original paper about the analysis of the UNION-FIND algorithm [39, p. 221]:

$$\alpha(m,n) := \min\{\, i \geq 1 \mid A(i, 4\lceil m/n \rceil) > \log_2 n \,\},$$

where $A(i,j)$ is a version of Ackermann's function that starts with $A(1,x) = 2^x$. The term $s\alpha(s,t)$ is thus dominated by the first two terms: For $s > t\log\log t$, we have $\alpha(s,t) = 1$, and $s\alpha(s,t) = O(s) = O(N)$. For $s \leq t\log\log t$, we have $s\alpha(s,t) \leq t\log\log t\,\alpha(s,t) \leq t\log\log t\,\alpha(t,t) = O(t\log t)$.

For the contour topology tree, one can apply the same analysis, except that one deals with a larger set of t' critical points.

Theorem 5 *The monotone path algorithm computes a contour tree in a triangulated d-dimensional mesh with N cells and t component-critical points in $O(N + t\log t)$ time and $O(N)$ space.*

The monotone path algorithm computes a contour topology tree in a triangulated two- or three-dimensional mesh with N cells and t' critical points in $O(N + t'\log t')$ time and $O(N)$ space. □

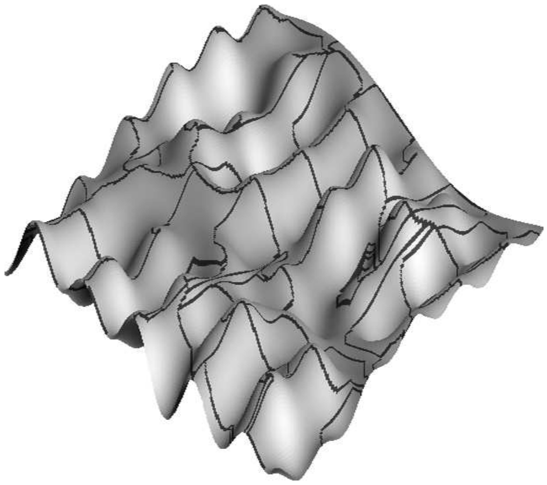


Fig. 3. Only a few points are visited.

Figure 3 shows a two-dimensional example with a grid of $n = 65536$ points. There are 75 join or split candidates, and only 5622 points are visited on monotone paths. The visited points are highlighted in black.

5.1 *Optimality*

The nodes in the contour tree are sorted along each monotone path in the tree. Bajaj *et al.* [3] have used this property to prove a lower bound of $\Omega(t \log t)$ on the worst-case time to construct the contour tree by a reduction from sorting. For any given set of t numbers, they showed how to construct a bivariate piecewise linear function on a triangulated domain of size $O(t)$, so that the contour tree contains a long monotone path from which the sorted sequence of t numbers can be read off. Together with the $\Omega(N)$ time for reading the input, this gives a lower bound of $\Omega(N + t \log t)$ on the running time for computing the contour tree. Therefore, our complexity of $O(N + t \log t)$ is *worst-case optimal.*

6 Experimental Results

We have implemented both our new monotone path algorithm presented in Section 4 and the sweep algorithm [8] described in Section 4.2 in C++/C, and ran our experiments on unstructured three-dimensional meshes. (Another set of independent implementation and experiments, programmed in Java and tested on structured three-dimensional meshes, was reported in the Master's thesis of Tobias Lenz [25].) The experiments are not extensive enough to draw a definite conclusion, but the results suggest that our algorithm compares favorably with the sweep algorithm [8].

We conducted our experiments on a Sun Blade 1000 workstation with 750MHz UltraSPARC III CPU and 4GB of main memory. The datasets are from real-world scientific visualization applications: The Blunt Fin (blunt, blunt2), the Liquid Oxygen Post (post, post2), and the Delta Wing (delta, delta2) datasets are from NASA, and the Combustion Chamber (comb, comb2) datasets are from Vtk [34] (generated from a combustion simulation). These datasets are all given as tetrahedral meshes, and each pair of datasets (e.g., blunt and blunt2) have the vertices sampled at different resolutions and hence their input sizes are different. Some representative isosurfaces generated from our experiments are given in Figures 4.

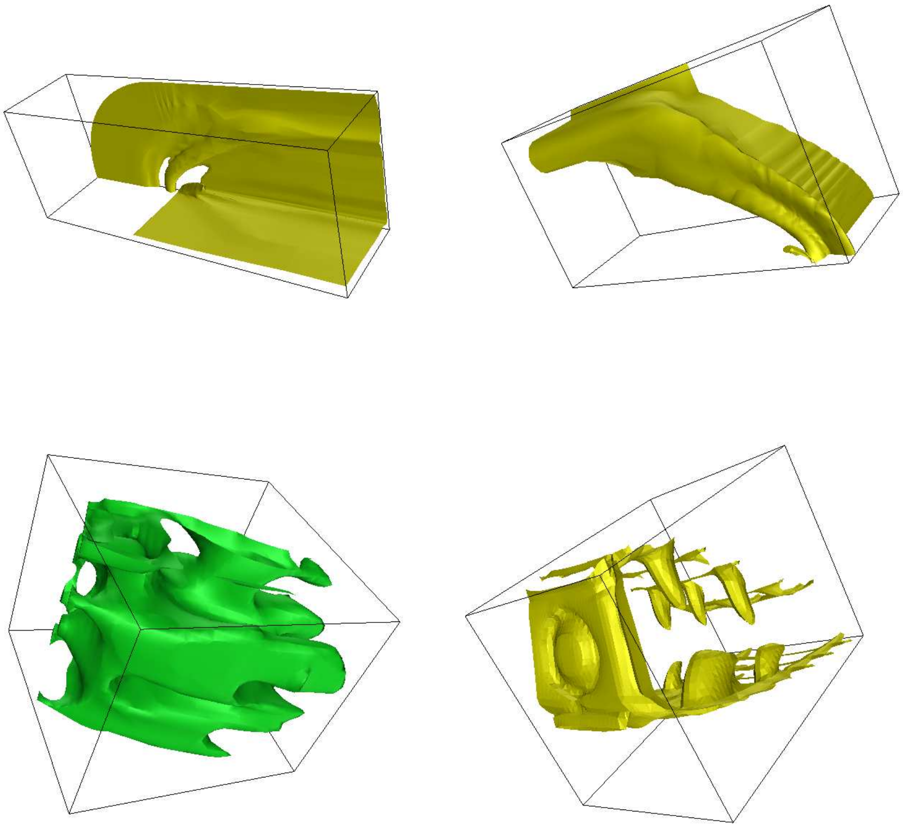


Fig. 4. Representative isosurfaces from our test datasets. The upper two are from the Blunt Fin dataset. The ones in the bottom are from the Combustion Chamber dataset.

Since the sweep algorithm [8] only computes the ordinary contour tree rather than the contour topology tree, for the purpose of comparisons we focused on computing the ordinary contour tree in the experiments. In our monotone path algorithm, we thus only need to consider the *component-critical points* rather than all critical points; recall from Section 4. Also, as described in Section 4, to

Table 3
Experimental results: some characteristics of the datasets. For each dataset, we list the number N of cells, the number n of vertices, the number t of *component-critical points*, the ratio t/n, the number s' of edges in the mesh with at least one component-critical endpoint, and the ratio s'/N.

Dataset	N	n	t	t/n	s'	s'/N
blunt	187395	40960	1820	4.44%	20072	10.7%
comb	215040	47025	524	1.11%	5791	2.7%
post	513375	109744	927	0.84%	10349	2.0%
delta	1005675	211680	1462	0.70%	17307	1.7%
blunt2	749580	228355	1833	0.80%	58159	7.8%
comb2	860160	262065	533	0.20%	13615	1.6%
post2	2053500	623119	947	0.15%	24241	1.2%
delta2	4022700	1217355	2042	0.17%	52913	1.3%

start a monotone path when constructing the join tree, it is sufficient to start from one representative of each of the C_- components of the lower neighbors, and similarly for the split tree. In our current implementation, for each "−" component we record the neighbor of the lowest function value and for each "+" component we record the neighbor of the highest function value. We do this for *every vertex*, component-critical or not, in the initial step of identifying component-critical points. The recorded neighbors can then also be used for quickly walking along a monotone path. We remark that the simplification of Section 4.4 is not implemented, and hence we need to sort all component-critical points, including all minima and maxima. However, as will be seen, the sorting step turned out to be very fast, and did not create a bottleneck of the running time.

In Table 3, we show some statistics of the datasets. It is very interesting to see that for all the datasets tested, the number t of *component-critical points* is typically fairly small, ranging from 0.15% to 4.44% of the input vertices. This certainly shows the importance of having an *output-sensitive* contour tree algorithm, where the size of the output contour tree is $O(t)$ rather than $O(n)$. The number s' of edges with at least one component-critical endpoint is less than 3% of the number N of cells for all but two datasets tested. Recall from Section 5 that $s = \sum_{i=1}^{t} C_-(v_i)$, where the sum is over all component-critical points v_i; clearly s' is a crude upper bound on s. Therefore $2s'$ is a crude upper bound on the total number of monotone paths started as well as the total number of FIND operations performed in computing *both* the join tree and the split tree. The overall UNION-FIND operations take time $O(s \cdot \alpha(s,t))$; with $\alpha(s,t)$ growing so slowly that it can be regarded as a constant no more than 4 for all practical purposes [12], we see that $s \cdot \alpha(s,t)$ is much smaller than

N, meaning that the time for performing the UNION-FIND operations is by far dominated by the initial $O(N)$-time scan. For a triangulated 3-manifold, the number m of edges can be roughly estimated as $m \approx n + N$, by using Euler's formula and neglecting the effect of the boundary faces.

Table 4
Experimental results: some statistics of the algorithms. For each dataset, we list the number of vertices (and the percentage) visited by our monotone path algorithm in computing the join tree (V_{join} and V_{join}/n) and in computing the split tree (V_{split} and V_{split}/n), respectively, as well as the total number of vertices visited in computing both trees by our algorithm (V_{path}) and by the sweep algorithm [8] (V_{sweep}), and their ratio in percentage.

Dataset	V_{join}	V_{join}/n	V_{split}	V_{split}/n	V_{path}	$V_{\text{sweep}} = 2n$	$\frac{V_{\text{path}}}{V_{\text{sweep}}}$
blunt	9958	24.3%	8345	20.4%	18303	81920	22.3%
comb	4953	10.5%	3257	6.9%	8210	94050	8.7%
post	12163	11.1%	4544	4.1%	16707	219488	7.6%
delta	13669	6.5%	10157	4.8%	23826	423360	5.6%
blunt2	9894	4.3%	8310	3.6%	18204	456710	4.0%
comb2	4802	1.8%	3180	1.2%	7982	524130	1.5%
post2	11927	1.9%	3906	0.6%	15833	1246238	1.3%
delta2	16875	1.4%	12028	1.0%	28903	2434710	1.2%

Table 4 shows the number of the vertices visited by our algorithm on the monotone paths during the construction of the join tree and the split tree. We see that the number of visited vertices is indeed very small, ranging from as low as 0.6% to 24.3% of n, showing that our algorithm only explores a small portion of the input, and the theoretical $O(m)$ time bound for exploring all monotone paths is only a pessimistic estimate. We also list the total number V_{path} and V_{sweep} of the visited vertices in computing *both* the join and split trees, respectively for our monotone path algorithm and the sweep algorithm of [8]. Note that the sweep algorithm performs a sweep over all vertices in computing each tree, and thus $V_{\text{sweep}} = 2n$. It can be seen that the ratio of $V_{\text{path}}/V_{\text{sweep}}$ ranges from 1.2% to 8.7% except for the blunt dataset (whose ratio is 22.3%), showing a very interesting and advantageous behavior of our algorithm.

In Table 5, we compare the running times of our algorithm with those of the sweep algorithm [8]. We give a break-up of the running times into the successive steps of the two algorithms: for the sweep algorithm, we list the times for sorting all vertices, computing the join and split trees, and the total running time; for our algorithm, we list the times for identifying the component-critical points, sorting all component-critical points, computing the join and

Table 5
Experimental results: running times. For each dataset, we list the running times (in seconds) for certain steps of the sweep algorithm [8] (sorting all vertices, computing the join and split trees, and the total time) and of our monotone path algorithm (identifying the component-critical points, sorting all component-critical points, computing the join and split trees, and the total time). The total times do not include the time for reading the input file from disk.

	Sweep (sec)			Path (sec)			
Dataset	Sort	J+S Trees	Total	Critical	Sort	J+S Trees	Total
blunt	0.08	1.43	1.51	1.28	0.01	0.02	1.31
comb	0.10	1.74	1.84	1.47	0.01	0.01	1.49
post	0.27	4.14	4.41	3.54	0.03	0.03	3.60
delta	0.53	8.54	9.07	6.94	0.06	0.06	7.06
blunt2	0.53	6.50	7.03	5.35	0.06	0.05	5.46
comb2	0.60	8.18	8.78	6.17	0.06	0.04	6.27
post2	1.48	19.68	21.16	14.79	0.14	0.09	15.02
delta2	4.30	41.85	46.15	28.90	0.31	0.19	29.40

split trees, and the total running time. The total running times exclude the time for reading the input from disk and the time for merging the join tree and the split tree. These steps are the same for both algorithms, and moreover, the time for merging the two trees is usually negligible.

It is interesting to see from Table 5 that the sorting step of the sweep algorithm, although contributing to the dominating term in the theoretical runtime complexity, ran very fast in practice, and that the sweeping operations for computing the join and split trees are by far the bottleneck steps for the sweep algorithm. Our algorithm, on the other hand, significantly reduced the running time of computing these two trees (the speed-up can be more than 220 times as fast!) so that it is no longer a bottleneck. This is due to the fact that typically our algorithm only visited 1.2%–8.7% of the vertices visited by the sweep algorithm, as seen in Table 4. Our algorithm also reduced the sorting time, as we only sort t component-critical vertices rather than all n vertices. This improvement is less crucial, however, since the sorting step is already very fast in the original sweep algorithm. It is important to observe that the above benefits of our algorithm have to be offset by the initial step of identifying the component-critical points, which is by far the major bottleneck of our algorithm. This bottleneck step is still faster than the bottleneck steps of the sweep algorithm (computing the join and split trees) in all tested datasets, and the speed difference increases as data size increases. Finally, comparing the total running times of the two algorithms, we see a clear advantage of our algorithm: our algorithm is always faster, and 1.57 times as fast in the best case. The savings in running time by our algorithm become more pronounced for larger data; for the largest dataset in Table 5, we reduce the running time from 46.15 seconds to 29.40 seconds.

7 Conclusions

7.1 Comparisons to Previous Algorithms

7.1.0.12 Theoretical Comparisons. The best previous algorithm takes time $O(N+n\log n)$. The new complexity $O(N+t\log t)$ is superior if $t \ll n$ and if $n\log n$ dominates N. For "nice" meshes, $N = O(n)$ and the last condition is always fulfilled. The size of t depends of course very much on the data. It is easy to construct a planar mesh with about $n/2$ saddle points and $n/4$ maxima and minima. However, if one thinks about reasonably smooth functions that are successively represented on a finer and finer grid, the number of critical points stays essentially constant and does not grow with n. When comparing the figures in Table 3 for different sampling rates (for example blunt and blunt2), one can see that the number of component-critical vertices is indeed independent of the resolution, for these examples. On the other hand, in cases of simulations of complicated physical phenomena like turbulence or in noisy data resulting from measurement, a large fraction of vertices may be critical.

7.1.0.13 Practical Issues for Large Datasets. From a practical point of view, sorting of data is not an issue in practice. Numeric data that come from real-world applications are usually reasonably well-behaved and can be sorted in linear expected time by bucketing techniques. *Radix sort* is a practical way to sort even arbitrary sets of `floats` or `doubles` in linear time. Thus, the theoretical $\Theta(n\log n)$ lower bound on sorting does not play a role in comparing different algorithms. Also, the experiments in Section 6 show that sorting is not the bottleneck.

However, for huge datasets, our algorithm has other advantages. Assume that the input data (the data points together with connectivity information) do not fit into main memory, but are kept in virtual memory or even in external memory (on a disk), or on a read-only medium (a CD-ROM).

In the initialization phase, the algorithm scans the data once and selects the critical points. This is a purely local operation if the neighboring cell information is available locally. Under the assumption that the critical points are only a small fraction of the data, they can be kept in main memory and sorted quickly. Walking along monotone paths is an operation that accesses the data in a spatially coherent fashion. If the layout of the data in its storage medium could be arranged so that it reflects spatial locality (points which are contiguous in space and which are connected in the mesh tend to be stored in adjacent memory cells), we may expect that our algorithm will perform well with today's hierarchical memory systems.

Contrast this with an algorithm that processes all points in sorted order. This will lead to a rather random access pattern, which limits the size of problems that can be treated to what can fit into main memory. The datasets for our experiments in Section 6 are too small to show any effect of this sort.

In our algorithm, we must mark the visited points. This is the only place where the original dataset is modified. However, even this can be avoided if the visited vertices on all monotone paths are still a small enough subset. Then we can keep a copy of the visited vertices in memory, storing them in a hash table, for example. Thus, all access to the original data is *read-only.*

If a deterministic rule for walking along monotone paths is chosen (see below), one can save more space by storing only every, say, 10th vertex of the monotone paths. This comes at a small expense in running time: a later path will continue for at most 10 extra steps after reaching a vertex that has already been visited. This is the same idea as described for the seed set method in Section 4.1.0.11.

7.2 Extensions and Future Work

7.2.0.14 Betti Numbers. In three dimensions, each edge of the contour topology tree can be augmented with information about the topology of the corresponding contour in the form of its Betti numbers $(\beta_0, \beta_1, \beta_2)$: β_0 is always 1; β_1 gives the number of independent tunnels, and β_2 is 0 or 1, depending on whether the contour is a surface with boundary or a surface without boundary. Pascucci and Cole-McLaughlin [32] gave an algorithm for this task which, however, requires the assignment of all vertices to edges of the contour topology tree and thus relies on the $O(n \log n)$-time sweep algorithm ($N = O(n)$ since they consider structured meshes) for computing the contour topology tree.

We are currently exploring whether we can compute topological information about the contours, like the Betti numbers, the genus, or the number of holes, within our framework, using only local information at the critical vertices. It would be interesting to compute this information about the upper and lower level sets as well.

7.2.0.15 Higher Dimensions. It should be possible to extend the characterization of critical points to 4-dimensional domains: a regular vertex ought to be a vertex v such that the level set through v dissects the link of v, which is a 3-dimensional sphere, into two 3-dimensional "half-spheres" (topological balls) meeting at a 2-dimensional sphere. It is easy to recognize when a 2-dimensional polyhedral manifold embedded in 3-space is a 2-sphere. By the three-dimensional Schönflies Theorem (see [1]), it follows that the two components into which the manifold cuts the link of v are homeomorphic to 3-balls. Thus, the proposed regularity condition should be easy to check.

We are not able to see any classification of regular and critical points in dimensions higher than 4. Intuition does not carry that far, and many properties which are true in low dimensions fail in higher dimensions. It is conceivable that such questions might be NP-hard or even undecidable.

7.2.0.16 Different Strategies for Monotone Paths. Our description of the monotone path algorithm leaves some freedom in the selection of the next outgoing edge in a monotone walk. There is a trade-off between several criteria. Our current implementation uses a steepest-descent-type selection rule: It always selects the vertex with highest or lowest value in the direction of the walk. Such a rule might lead to paths that achieve a big height difference and reach the bottom quickly, and to small seed sets. On the other hand, it may lead to a path that runs for a long time straight "in parallel" to an existing older path, where a movement to the side would lead to quick termination.

Other heuristics, like the first edge in cyclic order (starting at some chosen starting edge), some combination of left-most and right-most (in the plane), or completely random selection might perform better in practice. Of course, all of these may depend on the characteristic of the data, and are open to experimental evaluation.

Acknowledgements

We thank Wolfgang Kühnel, Ulrich Brehm, and Boris Aronov for discussions about critical points, and several anonymous referees for their critical and helpful comments.

References

[1] Alexander, J. W. On the subdivision of 3-space by a polyhedron. *Proc. Nat.*

Acad. Sci. (USA) **10** (1924), 6–8.

[2] Bajaj, C., Pascucci, V., and Schikore, D. The contour spectrum. In *Proc. IEEE Visualization* (1997), pp. 167–175.

[3] Bajaj, C., van Kreveld, M., van Oostrum, R., Pascucci, V., and Schikore, D. R. Contour trees and small seed sets for isosurface traversal. Tech. Rep. UU-CS-1998-25, Department of Computer Science, Utrecht University, 1998.

[4] Banchoff, T. F. Critical points and curvature for embedded polyhedra. *J. Diff. Geom.* **1** (1967), 245–256.

[5] Banchoff, T. F. Critical points and curvature for embedded polyhedral surfaces. *Amer. Math. Monthly* **77** (1970), 475–485.

[6] Brehm, U., and Kühnel, W. Combinatorial manifolds with few vertices. *Topology* **26** (1987), 465–473.

[7] Carr, H., and Snoeyink, J. Path seeds and flexible isosurfaces—using topology for exploratory visualization. In *Proc. Joint Eurographics-IEEE TCVG Sympos. Visualization (VisSym '03)*, G.-P. Bonneau, S. Hahmann, and C. D. Hansen, Eds. Eurographics Association, 2003, pp. 49–58 and 285.

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

[9] Chiang, Y.-J., and Lu, X. Progressive simplification of tetrahedral meshes preserving all isosurface topologies. *Computer Graphics Forum (Special Issue for Eurographics '03)* **22**, 3 (2003), 493–504.

[10] Chiang, Y.-J., and Lu, X. Simple and optimal output-sensitive computation of contour trees. Tech. Rep. TR-CIS-2003-02, Department of Computer and Information Science, Polytechnic University, 2003.

[11] Cole-McLaughlin, K., Edelsbrunner, H., Harer, J., Natarajan, V., and Pascucci, V. Loops in Reeb graphs of 2-manifolds. In *Proc. 19th Ann. Sympos. Comput. Geom.* (2003), ACM Press, pp. 344–350.

[12] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*, 2nd ed. MIT Press, Cambridge, MA, 2001.

[13] Cox, J., Karron, D. B., and Ferdous, N. Topological zone organization of scalar volume data. *Journal of Mathematical Imaging and Vision* **18** (2003), 95–117.

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

[15] Edelsbrunner, H., Harer, J., and Zomorodian, A. Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds. *Discr. Comput. Geom.* **30** (2003), 87–107.

[16] Fomenko, A. T., and Kunii, T. L., Eds. *Topological Modeling for Visualization.* Springer-Verlag, 1997.

[17] Gerstner, T., and Pajarola, R. Topology preserving and controlled topology simplifying multiresolution isosurface extraction. In *Proc. IEEE Visualization* (2000), pp. 259–266.

[18] Glaser, L. C. *Geometrical Combinatorial Topology*, Vol. I. van Nostrand Reinhold, New York, 1970.

[19] He, T., Hong, L., Varshney, A., and Wang, S. Controlled topology simplification. *IEEE Transactions on Visualization and Computer Graphics* **2**, 2 (June 1996), 171–184.

[20] Kettner, L., Rossignac, J., and Snoeyink, J. The Safari interface for visualizing time-dependent volume data using iso-surfaces and contour spectra. *Computational Geometry Theory and Applications* **25** (2003), 97–116.

[21] Kettner, L., and Snoeyink, J. A prototype system for visualizing time-dependent volume data. In *Proc. 17th Ann. Symp. Computational Geometry* (2001), ACM Press, pp. 327–328.

[22] Kosinski, A. Singularities of piecewise linear mappings, I. Mappings into the real line. *Bull. Amer. Math. Soc.* **68** (1962), 110–114.

[23] Kühnel, W. Triangulations of manifolds with few vertices. In *Advances in Differential Geometry and Topology*, F. Triceri, Ed. World Scientific, 1990, pp. 59–114.

[24] Kuiper, N. H. Morse relations for curvature and tightness. In *Proc. Liverpool Singularities Symposium II* (1971), C. T. C. Wall, Ed., Lecture Notes in Mathematics, Vol. 209, Springer-Verlag, pp. 77–89.

[25] Lenz, T. Efficient construction of contour trees using monotonic paths. Diplomarbeit (M.Sc. thesis), Institut für Informatik, Freie Universität Berlin, Aug. 2002.

[26] Miller, G. L., Teng, S.-H., Thurston, W., and Vavasis, S. A. Automatic mesh partitioning. In *Sparse Matrix Computations: Graph Theory Issues and Algorithms* (1993), A. George, J. R. Gilbert, and J. W.-H. Liu, Eds., Institute for Mathematics and its Applications, Springer-Verlag, pp. 57–84.

[27] Miller, G. L., Teng, S.-H., Thurston, W., and Vavasis, S. A. Geometric separators for finite-element meshes. *SIAM J. Sci. Comput.* **19** (1995), 364–386.

[28] Milnor, J. W. *Morse Theory.* Princeton University Press, Princeton, 1963.

[29] Morse, M. Topologically non-degenerate functions on a compact n-manifold. *J. Analyse Math.* **7** (1959), 189–208.

[30] Pascucci, V. On the topology of the level sets of a scalar field. In *Proc. 13th Canad. Conf. Comput. Geom. (CCCG 2001)* (University of Waterloo, Ontario, Canada, Aug. 2001), T. Biedl, Ed., pp. 141–144. `http://compgeo.math.uwaterloo.ca/~cccg01/proceedings/`.

[31] Pascucci, V. On the topology of the level sets of a scalar field. Tech. Rep. UCRL-JC-142262, Lawrence Livermore National Laboratory, Feb 2001.

[32] Pascucci, V., and Cole-McLaughlin, K. Efficient computation of the topology of level sets. In *Proc. IEEE Visualization* (2002), pp. 187–194. Full version to appear in *Algorithmica*.

[33] Schroeder, W., Lorensen, W., and Linthicum, C. Implicit modeling of swept surfaces and volumes. In *IEEE Visualization '94* (Oct. 1994), pp. 40–45.

[34] Schroeder, W., Martin, K., and Lorensen, W. *The Visualization Toolkit.* Prentice-Hall, 1996.

[35] Shinagawa, Y., Kunii, T. L., and Kergosien, Y. L. Surface coding based on Morse theory. *IEEE Comput. Graph. Appl.* **11** (Sept. 1991), 66–78.

[36] Takahashi, S., Ikeda, T., Shinagawa, Y., Kunii, T. L., and Ueda, M. Algorithms for extracting correct critical points and constructing topological graphs from discrete geographical elevation data. *Computer Graphics Forum* **14**, 4 (1995), C181–C192.

[37] Takahashi, S., Takeshima, Y., and Fujishiro, I. Topological Volume Skeletonization and its Application to Transfer Function Design. Tech. Rep. OCHA-IS 2000-3, Department of Information Sciences, Faculty of Science, Ochanomizu University, Ochanomizu, Japan, Feb. 2001.

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

[39] Tarjan, R. E. Efficiency of a good but not linear set union algorithm. *J. Assoc. Comput. Mach.* **22** (1975), 215–225.

[40] Tutte, W. T. How to draw a graph. *Proc. London Math. Soc., III Ser.* **13**, 52 (1963), 743–768.

[41] van Kreveld, M., van Oostrum, R., Bajaj, C. L., Pascucci, V., and Schikore, D. Contour trees and small seed sets for isosurface traversal. In *Proc. 13th Ann. Sympos. Comput. Geom.* (1997), pp. 212–220.

A Characterization of Critical Vertices

In this section we prove Theorems 1 and 2. It seems that such a basic statement should be well-known, but we could not find in the literature an explicit statement with a satisfactory proof.

Morse Theory for piecewise linear functions has been treated in Brehm and Kühnel [6, Section 2]; see also Kühnel [23, Chapter 7] for a more detailed account. Critical points are defined and the topology changes at these points are analyzed at the level of homology. For our case of two and three dimensions, this implies that regular points, where the homology is trivial, do not incur a topology change when the level set passes them, and there is a piecewise linear homeomorphism between different level sets [18]. This leads to the easy characterization of critical points in Theorem 1.

Our proof given below may still have merits since it constructs the isotopy quite explicitly. This might even be interesting for applications, for example when an isosurface is labeled by some characteristic pattern, which should be preserved as the surface is morphed across a non-critical point.

Apparently, Theorem 1 is so obvious that it is often taken for granted and not even *stated*. The fundamental work of Banchoff [4], who introduced Morse theory for piecewise linear functions, does not even define critical points (although it contains a *Critical Point Theorem*). Banchoff's more popular account [5] of the two-dimensional case defines critical points by the criterion in Theorem 3, but the connection to topology is not made.

Interestingly, in Morse theory for *continuous* functions [24,29], the criterion for the *definition* of a regular point v is just a local version of the *conclusion* of our Theorem 2: the existence of an isotopy between level sets in the neighborhood of v, i. e., some height-preserving homeomorphism between some neighborhood of v and the Cartesian product of a manifold with an interval of height values (such as the "cylinder" in Figure A.5).

A recent paper by Cox *et al.* [13] contains the statement of Theorem 1 for the interior vertices of the domain: the characterization of regular vertices by the condition $C_+ = C_- = 1$. The proof proceeds at a very high level and is not very detailed. The paper also claims (and proves) that the same characterization works in higher dimensions. This is not true. The standard smooth counterexample is the function $x^2 + y^2 - u^2 - v^2$ of four variables which has a critical point of Morse index 2 at the origin. A piecewise linear version of this example is given by the 4-dimensional cross-polytope formed as the convex hull of the eight points $(\pm 1, 0, 0, 0)$, $(0, \pm 1, 0, 0)$, $(0, 0, \pm 1, 0)$, $(0, 0, 0, \pm 1)$; see [22]. We add the origin as a vertex and triangulate the polytope by connecting the origin to each of the 16 facets, giving rise to one simplex in each orthant. The piecewise linear function defined by the values $f(0, 0, 0, 0) = 0$, $f(\pm 1, 0, 0, 0) = f(0, \pm 1, 0, 0) = 1$, $f(0, 0, \pm 1, 0) = f(0, 0, 0, \pm 1) = -1$ has $C_+ = C_- = 1$ at the origin, although the topology of $M_{\leq h}$ changes at $h = 0$. (In this example, interestingly, the level sets $M_{=+\varepsilon}$ and $M_{=-\varepsilon}$ are *homeomorphic* for all ε, by symmetry. This homeomorphism does not extend to an isotopy across $h = 0$, and it can be destroyed by appropriately extending M around the cross-polytope.) To achieve general position, the equal values $f(v) = 1$ and $f(v) = -1$ at the vertices would have to be slightly perturbed, but this does not affect the situation at the origin.

A.0.0.1 Proof of Theorem 2. Let us first assume that no vertex has a value in the range $[a, b]$. We can construct the isotopy separately in each tetrahedron T of M through which the level set $M_{=h}$ passes. Let T_+ be the convex hull of the vertices of T with high values (higher than b), and let T_- be the convex hull of the vertices with low values. In three dimensions, there are two possibilities: either one of these sets is a point and the other is a triangle (Figure A.1a) and the level sets are triangles, or we have two line segments and the level sets are quadrilaterals inside T (Figure A.1b). Every point of T lies on a unique line between a point in T_+ and a point in T_-. (This is true in any dimension.) This defines a canonical mapping between different level sets that pass between T_+ and T_-. For different tetrahedra which share a face, the mapping coincides on the common face and defines an isotopy between all level sets in the range $[a, b]$.

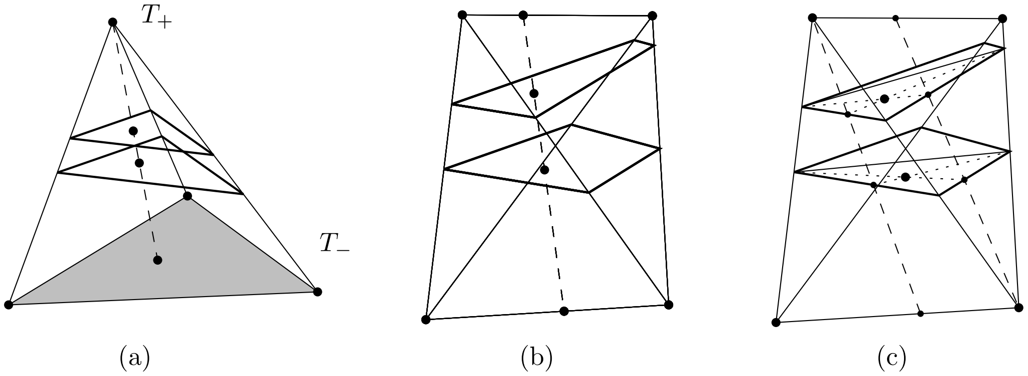


Fig. A.1. (a) The canonical mapping between different level sets by projection from A, the unique vertex of T_+. (b) The non-linear canonical mapping when $|T_+| = |T_-| = 2$. (c) Each quadrilateral is decomposed into two triangles, and the mapping is defined separately for each triangle.

Note however, that this mapping is in general not linear when T_+ and T_- are line segments, as in the case of Figure A.1b. To make it linear, we can arbitrarily select a diagonal which triangulates the quadrilateral and define the mapping by linear interpolation, see Figure A.1c. The mapping is unchanged at the boundary of the tetrahedron, and thus it is still smooth across different tetrahedra.

Let us now consider the case that only a single vertex v has a value in the range $[a, b]$. A described above, we can construct the isotopy inside all tetrahedra which do not contain this vertex. To define the isotopy inside the tetrahedra which contain the vertex v, we have to look at the structure of its neighborhood. The *star* S of v consists of all cells which contain v. This is illustrated for a two-dimensional manifold in Figure A.2. Alternatively, this figure can be interpreted as a cross-section through a three-dimensional neighborhood of v.

Let v be an interior point of the three-dimensional manifold M. The *link* L of v can be obtained by intersecting a small sphere centered at v with M. It

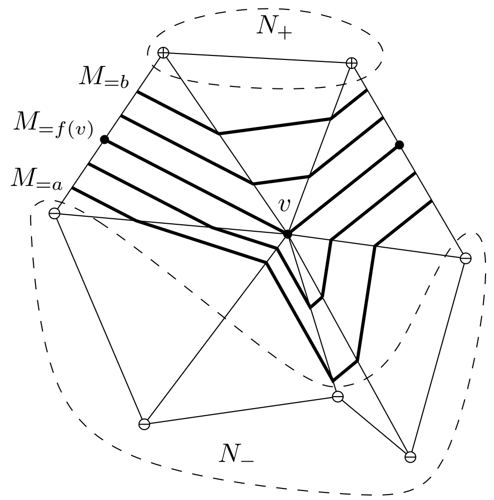


Fig. A.2. The star S and the neighborhood of a vertex v in two dimensions, and a few level sets above and below v.

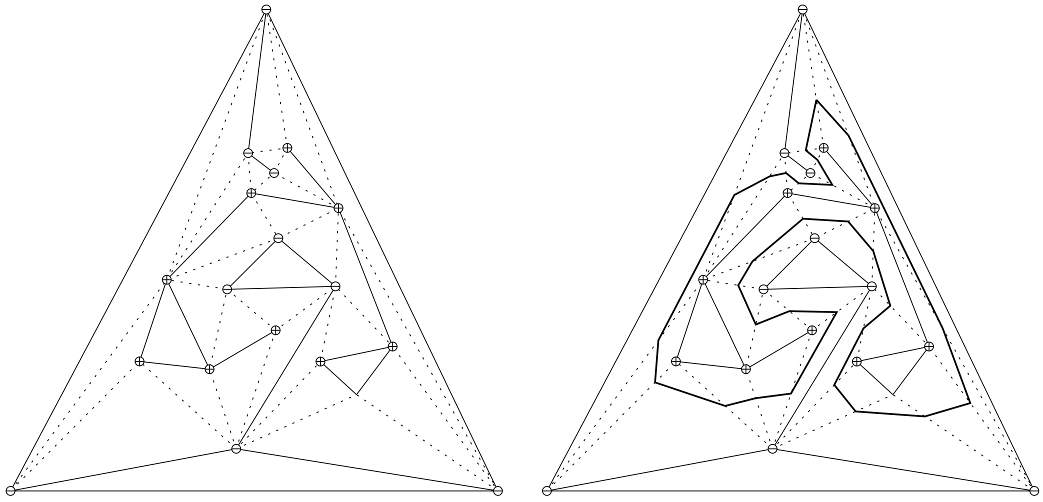
has the structure of a triangulated 2-sphere. Its vertices, edges, and triangles come from the tetrahedra, triangles, and edges of M which are incident to M. The neighborhood $N(v)$ is the underlying graph of L.

Figure A.3a shows an example of a neighborhood as a plane graph. Let us look at the one-dimensional manifold X in which the level set $M_{=f(v)}$ intersects the link. (In Figure A.2, X is represented by the two dots on the level set through v.) Every triangle of L which contains both a vertex of $N_+(v)$ and a vertex from $N_-(v)$ contributes an edge to X, and every edge of L which connects $N_+(v)$ with $N_-(v)$ contributes a vertex to X, see Figure A.3b. It follows that X consists of disjoint cycles. These cycles separate $N_+(v)$ from $N_-(v)$.

Now, if v is a regular point, then $N_+(v)$ and $N_-(v)$ each form a single connected component, and hence X is a single cycle. X cuts the triangles between $N_+(v)$ and $N_-(v)$ into triangles and quadrilaterals.

For $h = f(v)$ the level set $M_{=h}$ inside S has the structure shown in Figure A.4e: A single cycle X connected to the central vertex v like a wheel. At the upper end the level set $M_{=b}$ inside S has the structure shown in Figure A.4a: It consists of the subgraph of $N(v)$ generated by N_+, and the triangles and quadrilaterals at the interface between N_+ and the boundary cycle. This boundary cycle has the same number of sides as X. As h varies from b to $f(v)$ the graph generated by N_+ shrinks gradually down to a point. All triangles and edges in N_+ shrink proportionally in size. The triangles and quadrilaterals at the boundary and the edges of the boundary change their size and shape.

We have to interpolate between the initial and the final situation through all intermediate situations. One option would be to proceed through a number of steps and construct the isotopy accordingly. For example, one can successively



(a) (b)

Fig. A.3. (a) The link and the neighborhood graph of v. Higher and lower vertices are marked with + and −. N_+ and N_- have only one component each, and the dotted lines run between these two components. Although the graph is drawn with straight lines, it is intended to represent a decomposition of the sphere about v into spherical triangles. (b) The isoline X.

collapse N_+ to a point. Or one can use a sequence of elementary subdivision operations (by inserting a new vertex into a triangle) and their inverse *welding* operations. (This process of transforming a triangulated version of the structure shown in Figure A.4a into the wheel of Figure A.4e is called *starring* [18, Theorem II.11].)

In contrast, we will use a "one-shot" approach to construct the isotopy. We start with the situation of Figure A.4a. We triangulate the quadrilateral faces (arbitrarily). Now we find a plane straight-line embedding this graph in which the outer face is a regular polygon P, as in Figure A.4b. By the theorem of Tutte [40], such a straight-line drawing exists.

The following construction is essentially the *Alexander trick*, which has been used to extend a piecewise isotopy from the boundary of two manifolds to their interior. We create the prism $Z = P \times [a, b]$, see Figure A.5. On the top face $P \times \{b\}$ we embed the triangulation of Figure A.4b that we have constructed. On the lower face $P \times \{a\}$ we embed an analogous triangulation for $M_{=a}$. The sides are formed by vertical quadrilaterals. Now we partition Z by connecting each face of the boundary with the central vertex in $P \times \{f(v)\}$, which is going to represent the vertex v. The typical intersection of Z at height h ($f(v) < h < b$) looks as in Figure A.4c: A shrunk copy of P connected to the outer polygon in the original size of P by a ring of quadrilaterals. We triangulate these quadrilaterals (arbitrarily) and obtain the triangulation in Figure A.4d: We have the piecewise linear mapping $g_1\colon M_{=b} \to P$ defined through the compatible triangulations of $M_{=b}$ and P (Figure A.4b). It is straightforward

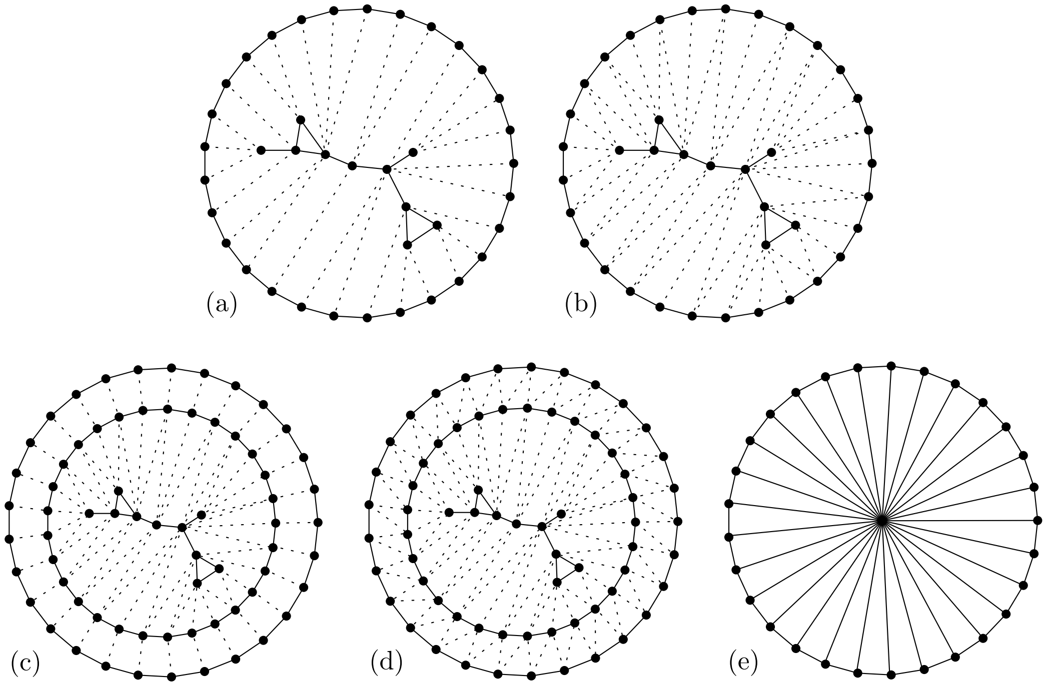


Fig. A.4. (a) The structure of the level set $M_{=b}$ in the cells incident to v, drawn as a plane graph. (b) The same graph, triangulated, is the top face of Z. (c) The same polygon with a shrunk copy of N_+. (d) The same graph, triangulated, is a horizontal section of Z at an intermediate level. (e) Horizontal section of Z at $f(v)$, having the same structure as the level set $M_{=f(v)}$ in the cells incident to v.

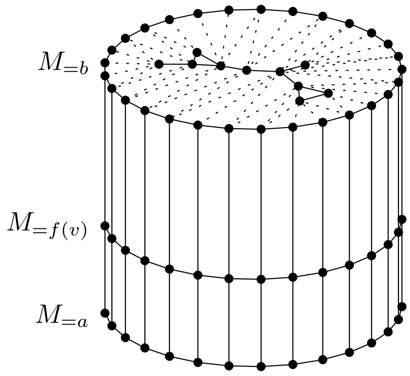


Fig. A.5. The prism Z.

to extend this to a mapping $g_2\colon M_{=b} \times [a, b] \to Z$ by $g_2(x, h) := (g_1(x), h)$. To get the desired isotopy we have to provide a mapping g_3 from Z to M respecting the heights, i.e., $f(g_3(y, h)) = h$.

Figure A.6 shows typical cases of the evolution of $M_{=h}$ inside a single tetrahedron $T = vABC$ as h varies from b to $f(v)$. The triangle ABC belongs to boundary of the star of v, and the mapping has to coincide with the mapping given for the tetrahedra which do not contain v. The points on ABC must be mapped to a quadrilateral side of Z. We concentrate on the upper part

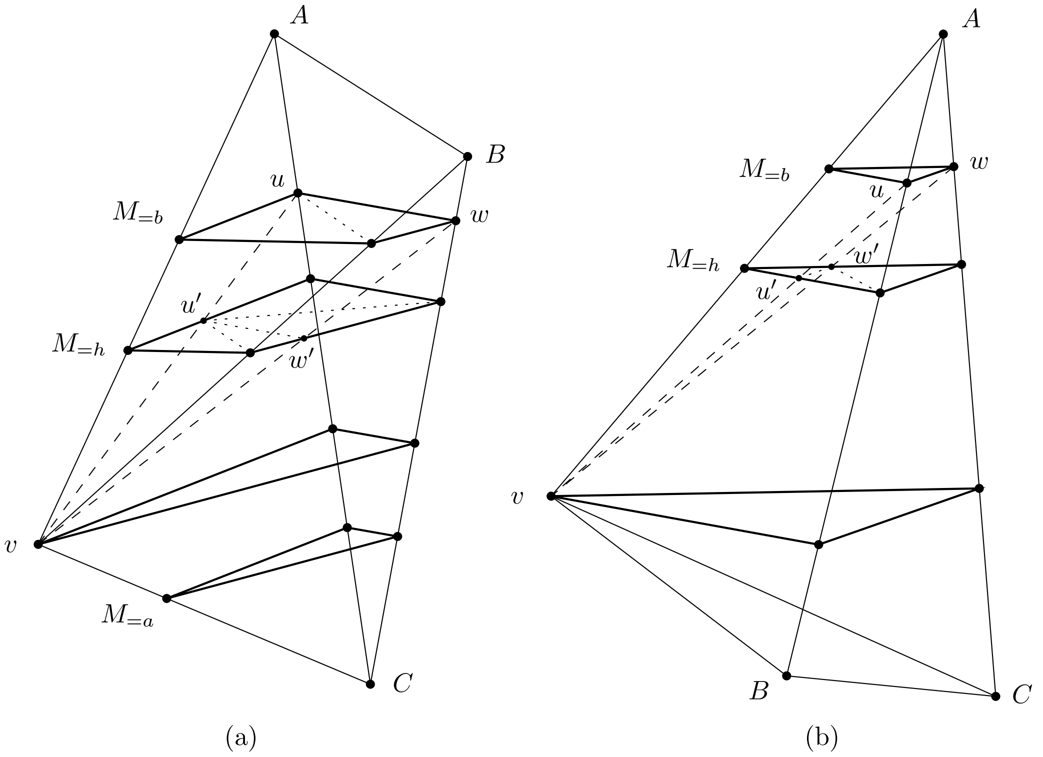


Fig. A.6. (a) The decomposition of a cut through a tetrahedron into four triangles when two vertices are higher than v. (b) The decomposition of a cut through a tetrahedron into three triangles when one vertex is higher than v.

$h \geq f(v)$.

In the case of Figure A.6a, when T has two vertices A and B which are higher than v and one vertex C below v, the intersections with $M_{=h}$ are quadrilaterals. The two edges AC and BC, which cross $M_{=f(v)}$, intersect $M_{=b}$ in two points u, w. They represent boundary vertices of Z on the top face. In each level set $M_{=h}$, we define points u' and w' on the lines vu and vw, and we use the segment $u'w'$ to partition the quadrilateral $M_{=h} \cap T$ into two quadrilaterals: a trapezoid and another quadrilateral which is a scaled copy of the quadrilateral in $M_{=b}$. These two quadrilaterals can be mapped to the corresponding pieces of $P \times \{h\} \subset Z$ (Figure A.4d) after triangulating them accordingly, as shown in Figure A.6a.

In the case of Figure A.6b, when T has one vertex higher than v and two vertices below v, the intersections with $M_{=h}$ are triangles. Similarly as above, we cut each triangle into a triangle and a trapezoid, and after triangulating the trapezoid, we can map everything to $P \times \{h\} \subset Z$.

The easy case when T has three vertices higher than v is not shown. In this case, the intersections with $M_{=h}$ are triangles which can be trivially mapped to the corresponding triangles of $N_+(v)$.

It is now easy to see that the mapping which has been defined is piecewise linear for each height $h \geq f(v)$, and it interpolates smoothly between $M_{=b}$ and $M_{=f(v)}$. The lower half, from $M_{=f(v)}$ to $M_{=a}$ can be treated similarly.

The above arguments can be adapted to the case when v lies at the boundary of M without difficulty.

This concludes the treatment of the case when $[a, b]$ contains a single vertex v. If several vertices have a value in this interval, we partition it into subintervals containing only a single vertex (this is possible by our assumption that all values $f(v)$ are distinct), and we connect the resulting isotopies. In fact, the argument extends to the case of vertices with equal values as long as these vertices are not adjacent. This concludes the proof of Theorem 2. $\square$

As mentioned above, the isotopy $g(x, h)$ is piecewise linear only as a function of x. It would be interesting to find a construction in which the "orbit" $g(x, h)$ of every point $x \in M_{=a}$ is also piecewise linear as a function of h.

1.0.0.2 Proof of Theorem 1. One direction of Theorem 1 is obtained as an easy corollary of Theorem 2: If v is not one of the critical points listed, then all level sets immediately above and below v are homeomorphic. The other direction is easy. $\square$