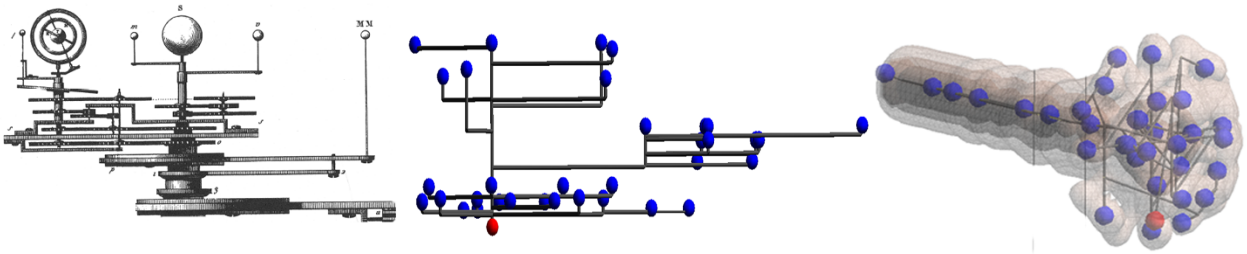
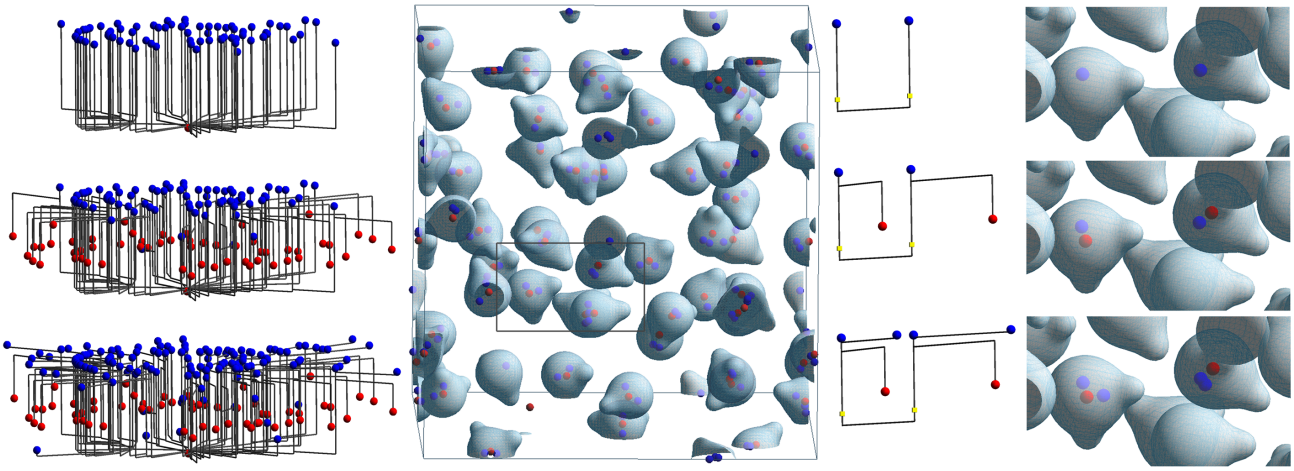
Multi-Resolution computation and presentation of Contour Trees *

V. Pascucci K. Cole-McLaughlin G. Scorzelli



(a) (left) Orrery reproducing the hierarchical relationship between the orbits of the sun, the planets and their moons. Original design (1812) by A. Janvier reprinted recently by E. Tufte [25]. (center) Contour Tree drawn as an orrery where stars/planets/moons are replaced by critical points and the nested orbits represent the hierarchy of topological features. The scalar field of this particular example is the inverse air density distribution α of a fuel injection into a combustion chamber. (right) Semi-transparent level sets of α displayed together with the Contour Tree drawn with its critical points in their original position.



(b) (left) Contour Tree at three levels of resolution for the electron density distribution (ρ) computed with an *ab initio* simulation for water molecules at high pressure. At the coarse scale (top) the tree has only one maximum (blue sphere) per water molecule. At medium scale (middle) the topology reconstructs one dipole per molecule with one maximum and one minimum (red sphere). At fine scale (bottom) only the noise is removed and three extrema per molecule reconstruct each atom in the simulation. (middle) Semi-transparent level sets of ρ together with the fine scale topology. Adaptive refinement in the area marked with a rectangle can be used to refine the topology for two particular molecules. This as shown on the right at coarse, medium and fine scale both for the Contour Tree and for the embedding (shown side by side).

Figure 1: Visualization of multi-resolution Contour Trees for volumetric scalar fields. The topology is visualized both embedded in the input domain (critical points in their original position) and in an abstract layout that presents topological features using the metaphor of an orrery.

ABSTRACT

The Contour Tree of a scalar field is the graph obtained by contracting all the connected components of the level sets of the field into points. This is a powerful abstraction for representing the structure of the field with explicit description of the topological changes of its level sets. It has proven effective as a data-structure for fast extraction of isosurfaces and its application has been advocated as a user interface component guiding interactive data exploration sessions. In practice, this use has been very limited due the problem of presenting a graph that may be overwhelming in size and in which a planar embedding may be confusing due to self-intersections. Topological simplification techniques have helped in relieving this problem since they allow reducing the size of the graph.

We present a multi-resolution data-structure for representing contour trees and an algorithm for its construction. Moreover, we provide a hierarchical layout that allows coarse-to-fine rendering of the tree in a progressive user interface.

Construction of our multi-resolution model is only slightly more expensive than the standard tree, but introduces far greater flexibility when filtering, both uniformly and adaptively, the topology of the data by importance with respect to different metrics. We have tested the approach using topological persistence (that is the difference in function value between a pair of critical points that are simplified) as the main metric for constructing the topological hierarchy, and using geometric position (containment in a bounding box) as a secondary metric for adaptive refinement.

*This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

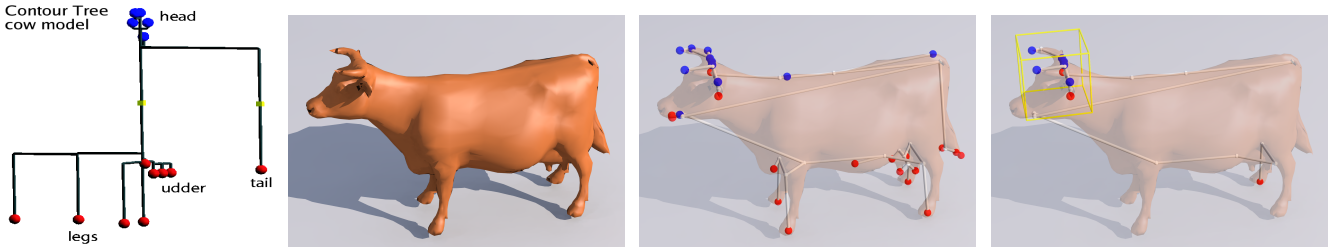


Figure 2: Topology of a triangulated cow model. (a) Contour Tree computed using as Morse function f the height in the vertical direction. (b) The original cow model. (c) Contour tree drawn with the critical points in their original position on the mesh. (d) Adaptive refinement of the Contour tree where the full topology is shown only in the bounding box containing the horns.

1 Introduction

A Morse function over a domain $\mathcal{D}$, is a smooth mapping, $f : \mathcal{D} \rightarrow \mathbb{R}$, such that all its critical points (maxima, minima and saddles) are distinct. Complex natural phenomena, both sampled and simulated, are often modeled as Morse functions. MRI scans generate Morse functions that are used in medical imaging to reconstruct human tissues. Electron density distributions computed by high-resolution molecular simulations are Morse functions whose topology express bonds among the atoms in molecular structures. The structure of geometric models used in computer graphics and CAD applications can be effectively represented in terms of the topology of a Morse function [14].

The Reeb graph [19] is a simple structure that summarizes the topology of a Morse function. For functions with simply connected domains this graph is also simply connected and is called the Contour Tree. The Reeb graph has been used to analyze the evolution of teeth contact interfaces in the chewing process [20], and to compute indices of topological similarity for databases of geometric models [14]. Topological information has been used to guide the construction of transfer function for volume rendering of scientific data [26, 22]. A more extensive discussion of the use of the Reeb graph and its variations in geometric modeling and visualization can be found in [12].

The first algorithm for constructing Reeb graphs of Morse functions with two-dimensional domains is due to [21]. Given a triangulated surface, this scheme takes as input the set of all distinct level lines and therefore has worst case time complexity $O(n^2)$, where n is the number of vertices in the triangulation. An $O(n \log n)$ algorithm for computing contour trees in any dimension was introduced in [6]. This scheme has been extended in three dimensions to include the genus of all isosurfaces [18]. The first multi-resolution representation of the Reeb graph was introduced in [14]. Their method hierarchically samples the range space of f while concurrently refining the Reeb graph. They obtain a multi-resolution model that is suitable for fast comparison of graphs. However, this hierarchy does not represent the topology of f at multiple levels of detail. A formal framework for ranking topological features by persistence has been introduced in [10] and applied to two-dimensional Morse functions in [2]. Topological simplification is used in [22] to design transfer function that highlight only the major features in the data. Topological simplification is also widely used in vector field visualization to highlight the most important structures present in the data [23, 24].

Integration of the Contour Tree in user interfaces to help selecting isosurfaces has been first suggested [1] but was only fully developed six years later in [3]. The latter work is particularly interesting for the use of a new concept of "Path Seeds" that links explicitly the arcs in the Contour Tree to distinct connected components (contours) of the level sets. This introduces the powerful new paradigm of selecting contours instead of entire isosurfaces. The most recent extensions of this work introduce high quality topological simplification [4] where the contour tree is pruned incrementally to reduce its complexity and highlight the fundamental structures present in the data. The scheme involves exact and approximate computation of several metrics that are used for ranking the "importance" of an arc before pruning. The results are extremely compelling since they highlight the direct correlation of anatomic parts in medical data with single branches in the contour tree. The only limitation is this approach is the lack of a real multi-resolution representation of the topological information. As it is well know in the computer graphics community, simplification schemes are best suited for computation of high quality approximations but tend too be inappropriate for computing real-time adaptive refinement for interactive data exploration. In this case we have the additional problem of needing to compute the layout for the graph, which may be very expensive when done with an external tool [16] not designed for real-time interaction. Consequently the layout is either poorly distributed at a coarse level, since it was optimized for a fine level of detail, or it is recomputed after simplification, which may change completely its layout loosing the correlation among different levels of resolution with a confusing effect for the user.

Our scheme employs a multi-resolution representation of the Contour Tree and an integrated 3D layout. In particular, we adapt a simple radial graph drawing algorithm [8] to achieve a 2D layout without self intersections. The arcs in the layout are then embedded in 3D by moving each node to a z elevation equal to the function value of the corresponding critical point. Each arc is mapped to an L shape that connects its end nodes at different elevations. The result is an orrery-like model (see Figure 1(a)), which is similar to other embeddings typically used for large scale hierarchies [15].

Contributions. Our results are summarized in Figures 1 and 2 with hierarchical computation and adaptive presentation of the topology of 2D and 3D scalar fields. The main three contributions of this paper are: (i) we provide a multi-resolution representation for the Contour Tree with algorithms for uniform and adaptive refinement on the basis of precomputed metrics; (ii) we provide an algorithm for computing a multi-resolution Contour Tree directly from join and split trees and guarantee that atomic simplification steps of the tree correspond to atomic collapses of proper pairs of critical points (the minimal topological simplification possible); (ii) we provide a simple scheme for laying out the tree in a way that highlights the hierarchical relationships among the critical points and that can be rendered progressively for real-time user interaction.

We leave to a longer version of this paper the formal proof of the topological correctness of our atomic simplifications. The practical results are demonstrated on datasets of different nature, such as surface meshes typically used in computer graphics and volumetric models representing scien-

tific data. More examples and a demo software can be found at `http://www.pascucci.org/topology/contour_tree`.

2 MULTI-RESOLUTION CONTOUR TREES

Let $\mathcal{D}$ be a triangulated domain and $f : \mathcal{D} \to \mathbb{R}$ be a function obtained by linear interpolation of the value of f at the vertices of $\mathcal{D}$. Morse theory provides a formal framework for understanding the topology of $\mathcal{D}$ by analyzing the function f. The fundamental tool in Morse theory is the characterization of each point of $\mathcal{D}$ as being either regular or critical.

We assume that $\mathcal{D}$ is a simplicial complex. Therefore, every k-cell c of $\mathcal{D}$ is the convex hull of $k+1$ vertices of $\mathcal{D}$. Moreover, a cell c' is a called *face* of c if its vertices are a subset of those of c. If $c \in \mathcal{D}$ then all its faces must be in $\mathcal{D}$. For a vertex $v \in \mathcal{D}$, its *link* Lk_v is the set of cells that do not contain v but that are faces of some cell containing v. Furthermore, the *lower link* of v, Lk_v^-, is the set of all cells in Lk_v that have only vertices with function value smaller than $f(v)$. The *upper link* Lk_v^+ is the set of cells in Lk_v that have only vertices with function value greater than $f(v)$.

Definition 1 *Let $\mathcal{D}$ be a triangulated manifold with boundary and $f : \mathcal{D} \to \mathbb{R}$ be a piecewise-linear function. A vertex $v \in \mathcal{D}$ is called* regular *if both Lk_v^- and Lk_v^+ have exactly one connected component. Otherwise v is called a* critical point *and $f(v)$ is called a* critical value.

To deal simply with degenerate cases we use standard symbolic perturbation [11] by sorting the vertices by function value and simply replace $f(v_i)$ with i in any comparison.

Definition 2 *A* level set *of f is the pre-image of a real value ω, $L_f(\omega) = f^{-1}(\omega)$. Given a level set, $L_f(\omega)$, we call a connected component of $L_f(\omega)$ a* contour.

Morse theory describes how the the topology of $L_f(\omega)$ changes as the isovalue ω changes. One of the main results states that if a and b are such that the range $[a, b]$ contains no critical values, then $L_f(\omega)$ is homeomorphic to $L_f(\nu)$ for all $\omega, \nu \in [a, b]$. This has deep implications on the structure of the Contour Tree.

Definition 3 *Consider the graph obtained by contracting to a point each contour of every level set of f. For a Morse function of a general domain $\mathcal{D}$ this graph is called the* Reeb graph *and can have a number of cycles depending on the topology of $\mathcal{D}$ [7]. However, if $\mathcal{D}$ is simply connected the* Reeb graph *is also simply connected and is called* Contour Tree.

From the definition it can be seen that the nodes of the contour tree correspond to critical points of f and are therefore associated with the relative critical value. Furthermore, nodes that correspond to extrema are leaf nodes, and nodes that correspond to saddle points must have degree three (or higher in degenerate cases).

Hierarchical Tree Representation We define a multi-resolution representation of the contour tree that allows linear time access to simplified representations of the topology. Typically finite graphs are represented as a list of nodes and a list of arcs, where each arc is defined as a node pair. Here we use an alternative *branch decomposition* where a *branch* is defined as a monotone path in the graph traversing a sequence of nodes with non-decreasing (or non-increasing) value of f. The first and last nodes in the sequence are called the endpoints of the branch. All other nodes are said to be interior to the branch. A set of branches is called a *branch decomposition* of a graph if every arc the graph appears in exactly one branch of the set. The standard representation of a graph satisfies this definition, where every branch is a single arc. We call this the trivial branch decomposition

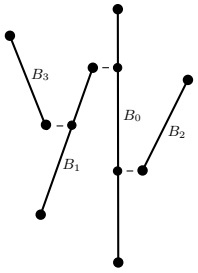


Figure 3: *Hierarchical decomposition* of a Contour Tree with four branches. The root branch B_0 connects the two global extrema. The branches B_2 and B_3 pair two maxima with split saddles and can be canceled independently. B_1 pairs a minimum with a join saddle and cannot be canceled before B_3 because of their parent-child relation.

Definition 4 *A branch decomposition of a tree is a* hierarchical decomposition *if: (i) there is exactly one branch connecting two leaves (called root branch), (ii) every other branch connects one leaf to an interior node of another branch.*

We construct a hierarchical decomposition of a contour tree such that the endpoints of each branch (except the root) represent a saddle-extremum pair that form an atomic topological cancellation (critical points can be canceled only in pairs). This is illustrated in Figure 3. The tree can be simplified by removing a branch that does not disconnect the tree. This corresponds to the cancellation of two critical points in the scalar filed. This simplification process defines a hierarchy of cancellations where a branch $B1$ is said to be the parent of branch $B3$ if one endpoint of $B3$ is interior to $B1$. The root branch has no parent and cannot be simplified. Removal of a parent before one of its children disconnects the tree. In the next section we will discuss the construction of a hierarchical decomposition based on the persistence of critical point pairs.

Once the hierarchy has been constructed we build different approximations of the original tree by incrementally connecting child branches to their parent starting from the root. To do so, we associate min-max values to each branch for several metrics (such as persistence, geometric location or other) and artificially enforce a nesting condition that requires the min-max value of the parent to contain the one of any child. This standard *saturation* of the ranges associated with each element of the hierarchy allows output sensitive construction of and adaptive refinement with a simple depth first traversal, without accessing any unnecessary element (see [13, 17] for more details).

Hierarchical Contour Trees Single resolution algorithms for computing a contour tree can be found in [6] and [18]. In both cases one needs to make two passes through the data to compute the *Join Tree* and the *Split Tree*. These trees are then merged to construct the Contour Tree. Here we use a similar approach but build directly a hierarchical decomposition of the Contour Tree. To do so we store all our trees as branch decompositions and modify the algorithm that merges the join and split trees. The general idea is illustrated in the example of Figure 4. At each stage (a row in the figure) the branches candidate for transfer into the Contour Tree are pointed by a gray arrow. The one effectively selected (by some measure of priority) is drawn with a bold and is the moved into the Contour Tree at the next stage. The large nodes are the endpoints of the branches (initially all). For example, in the second row, after the branch 10-6 has been moved to the Contour Tree, the Join Tree has a branch with endpoints 9 and 4 and middle node 6. Note in the third stage the red arrow that points to the branch 1-3-4, which is not a candidate for removal since the node 4 is properly paired with node 9 and not with node 1. At the fifth step the Join and Merge tree have been reduced to the branch 7-5-4-3-1, which becomes the root branch of the Contour Tree,.

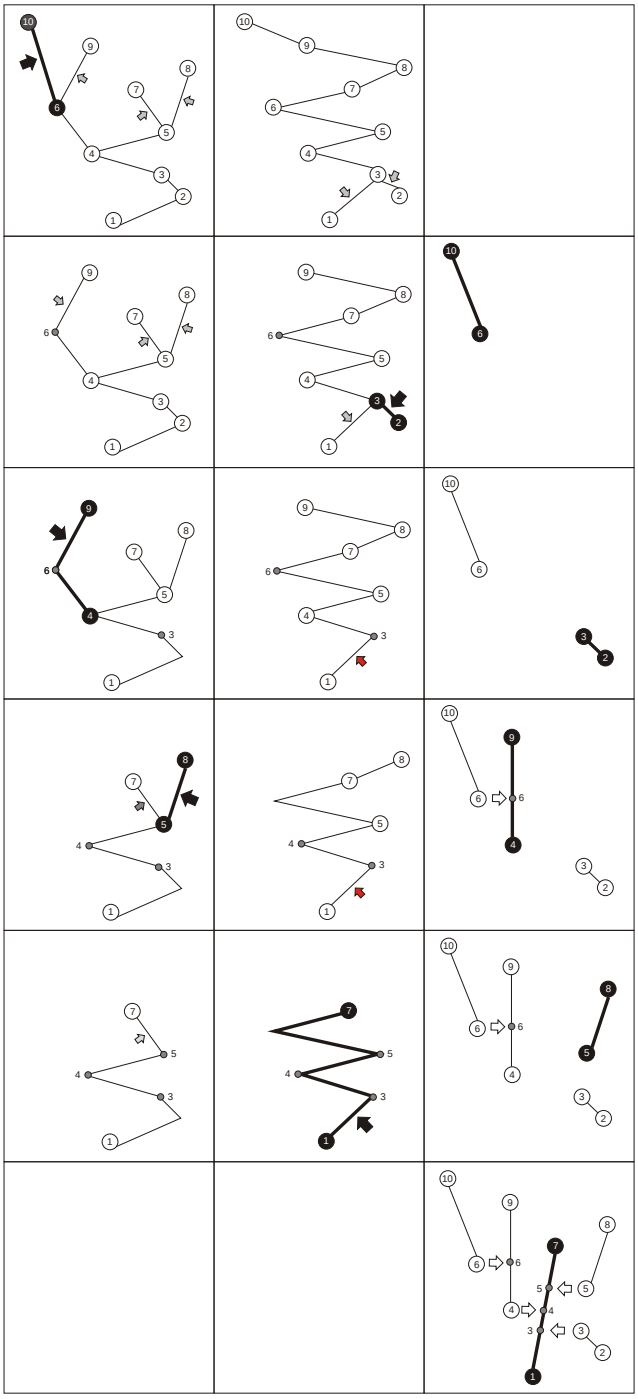


Figure 4: Incremental construction (top to bottom) of a hierarchical decomposition of a Contour Tree from its Join Tree and Split Tree. Each row of the table show all three trees: Join Tree on the left, Split Tree in the middles, and Contour Tree on the right.

Our data structure is based on branches, where the *length* of a branch is the difference in function value of its endpoints, taken in absolute value. This value is returned by Length(B). Leaf nodes are classified as either minima or maxima, by checking if the node is a starting node or ending node respectively. Furthermore, saddles are either join or split points. An interior node is a join saddle if it is the ending point of some branch, whereas it is a split saddle if it is the starting point of some branch. In this characterization a join saddle corresponds to a saddle point of f where two contours merge, and a split saddle to a saddle where one contour divides.

The function CanSimplify(G, B) returns *true* if the branch B in the graph G represents a valid cancellation. The first criterion for this to be true is that the branch must have no children. The second criterion that must be checked is that the endpoints of B are either a minimum and a join saddle or a split saddle and a maximum. In other words, a branch can be simplified if it connects directly two critical points that can be canceled in pair. In the long version of this paper we prove, for scalar fields of any dimension, that this selection always leads to a proper pairing of critical points in a topological sense.

Once a tree is constructed we can perform several queries on it. First, we include the function GetTree(B) that returns the tree that contains the branch B. For an arbitrary branch decomposition it is possible to have end nodes of degree two. We can check if a node, N, has degree two with the function IsRegular(T, N). If a node is a starting point we can perform the query UpBranch(T, N), which returns the branch that starts at the node. Likewise, we can call DownBranch(T, N) on ending points to access the branch that ends at the node. If CanSimplify(B) returns true for a branch B, then exactly one of its endpoints represents a saddle point. In this case we can access the unique saddle point of the branch by calling GetSaddle(B). Finally, a branch is defined to be a leaf branch if it has no interior nodes and one of its endpoints is a leaf node. The function IsLeafBranch(T, E) returns true if E is a leaf branch.

Join and Split Trees. Any of the standard algorithms for computing the join and split trees can be implemented, but the resulting trees must be stored as trivial branch decompositions. In these algorithms every node in each tree represents a critical point. Thus there will be some degree-two nodes, which correspond to saddle points from the other tree.

For completeness we briefly describe the algorithm for constructing the join and split trees that given in [6], which is a union-find sweep through the data that maintains in a graph the history of the union events. We make use of a standard Union-Find data structures that is managed by the functions NewUF(), NewSet(UF, i), Find(UF, i), and Union(UF, i, j), which respectively create the data structure, add a new set, return the set containing a given index, and merge two sets. The boolean functions IsMin(v) and IsCritical(v) return true if v is a local minimum of f and a critical point of f.

Algorithm 1. JoinTree

Input: Sorted array of n vertices ($\{v_i\}$) and a triangulated surface ($\mathcal{D}$).

Output: Join tree (JT).

1. $JT =$ NewGraph()
2. $UF =$ NewUF()
3. ***for*** $i = 0$ ***to*** $n - 1$ ***do:***
4. ***if*** IsCritical(v_i) ***then*** AddNode(JT, i)
5. $i' =$ NewSet(UF, i)
6. ***for each edge*** $v_i v_j$ ***with*** $j < i$ ***do:***
7. $j' =$ Find(UF, j)
8. ***if*** $j' \neq i'$ ***then*** AddBranch(JT, j', i')
9. Union(UF, i', j')
10. ***return*** JT

The algorithm for constructing the split tree, ST, is symmetric with and inverse sweep of the data points.

Multi-Resolution Contour Tree. In previous contour tree algorithms the Contour Tree, CT, is constructed form the JT and ST by "peeling off" leaves of the JT and ST and adding them to the CT. This approach uses a queue to store the leaves of the

JT and ST, which can be removed in any order. in our algorithm we impose instead a particular order of cancellation using a priority queue that provides always access to the shortest leaf branch. Once a branch is removed from the queue the adjacent branches in the JT and ST are merged, which is why JT and ST must be stored as branch decompositions. These merges can change the length (and therefore the priority) of the branches in the queue. The corresponding updated is done in a lazy way (nodes are updated only when they pop out of the queue) by the PopValid(PQ) function.

The priority queue is a standard data-structure that uses the operations: Pop(PQ) and Push(PQ, B), that retrieve the top element of the queue, and push a branch onto the queue respectively. It also supports the test IsEmpty(PQ) that returns true if there are no elements in the queue. In our case the priority is the length of a branch, B, and Pop(B) is guaranteed to return the branch with the lowest priority. The priority of a branch, B, can be queried using the function Priority(B).

Algorithm 2. PopValid
Input: Priority Queue (PQ) of branches
Output: Branch (B) that is a valid simplification

1. $B = \text{Pop}(PQ)$
2. $isValid = false$
3. ***while not*** $isValid$ ***do:***
4. ***if not*** CanSimplify(B) ***then:***
5. $B = \text{Pop}(PQ)$
6. ***else:*** ***if*** Length(B) $\neq$ Priority(B) ***then:***
7. Priority(B) = Length(B)
8. Push(PQ, B)
9. $B = \text{Pop}(PQ)$
10. ***else:*** $isValid = true$
11. ***return*** B

The procedure PopValid(PQ) ensures that we can pull the first branch that represents a valid cancellation from the queue. In this way we can ensure that each branch represents a topological simplification of f. For readability we introduce another subroutine of the contour tree algorithm that does the work of "peeling off" a leaf branch. In this routine we make use of the function MergeBranches(B_1,B_2) that merges the branches B_1 and B_2 into B_3.

Algorithm 3. PeelOffBranch
Input: Branch (B), Join Tree (JT) and Split Tree (ST)
Output: A branch representing a valid simplification or $null$.

1. $XT, YT, N = \text{MyTree}(B), \text{OtherTree}(B), \text{GetSaddle}(B)$
2. RemoveBranch(XT, B)
3. ***if*** IsRegular(YT, N) ***then:***
4. $B_1, B_2 = \text{UpBranch}(YT, N), \text{DownBranch}(YT, N)$
5. $B_3 = \text{MergeBranches}(B_1, B_2)$
6. ***if*** CanSimplify(B_3) ***then: return*** B_3
7. ***return*** $null$

Using the subroutines PopValid(PQ) and PeelOffBranch(B, JT, ST) the code for our main algorithm, BuildContourTree(JT, ST), is simple.

Algorithm 4. BuildContourTree
Input: Join Tree (JT) and Split Tree (ST)
Output: Contour Tree (CT)

1. $CT = \text{NewGraph};\quad PQ = \text{NewPQ}$
2. ***for each*** $B \in JT, ST$ ***do:***
3. ***if*** IsLeafBranch(B) ***and*** CanSimplify(B) ***then:***
4. Push(PQ, B)
5. ***while not*** IsEmpty(PQ) ***do:***
6. $B_{top} = \text{PopValid}(PQ)$
7. AddBranch(CT, B_{top})
8. $B_{next} = \text{PeelOffBranch}(B_{top}, JT, ST)$
9. ***if not*** $B_{next} \neq null$ ***then*** Push(PQ, B_{next})
10. ***return*** CT

It is clear from the discussion that this algorithm produces a multi-resolution contour tree, such that each branch represents a valid topological simplification. We can now define an order on the branches that allows to extract a contour tree after any number of simplifications in linear time. First, we define the persistence of a branch to be the greater of its length and the persistence of each of it children. This definition differs from the definition of persistence given in [10] because it takes into consideration the topological obstructions. Thus a pair of critical points is never assigned a persistence value that is less than any of its obstructions.

The same analysis as in [5] can be used to show that the complexity of BuildContourTree is $O(n \log n)$ where n is the number of nodes in JT and ST. Using a simple FIFO queue instead of a priority queue would yield a complexity of $O(n)$ but with the risk of building an unbalanced tree. In practice this does not seem to be a problem and a linear queue may be advisable for a faster implementation with lighter data-structures.

3 Layout, Traversal and Visualization

In a user interface that includes the Contour Tree as a tool for interactive exploration of topology, it is often desirable to use a tree layout that highlights naturally: (i) the separation among branches (either by movement or by minimization of self-intersections); (ii) the function value of the critical points (e.g. mapping it to the vertical direction in the layout); (iii) the *scale* of the topological features (e.g. level of persistence); (iv) hierarchical parent-child relations among topological features. Moreover, the layout should: (v) remain stable during the interactive navigation with changing adaptive refinement, and (vi) the information should be spread as uniformly as possible to optimize the use of the screen space.

As often happens, such objectives are often in conflict. For example a 2D layout using one axis to represent the function value of the critical points does not allow to guarantee a layout without self intersections. This problem is shown in Figure 7 where a simple terrain with very few extrema cannot be drawn without self intersections if one uses the planar embedding proposed in [1]. Because of this problem we develop a three-dimensional embedding of the contour tree, which is inspired to the orrery metaphor shown in Figure 1(a). This is illustrated by redrawing the graph of Figure 3 as in Figure 8(a), where each the root is a vertical line. All the other branches are L shapes that connect to their extremum (red for minimum, blue for maximum) to its paired saddle along the parent (white circle). We map the field value to the elevation along the z-coordinate so that the vertical span of each branch equals its persistence. The projection of the tree in the xy plane is arranged with a circular layout that has no self intersections as discussed below and illustrated in Figure 9.

Our visualization scheme makes use of an algorithm for the layout of rooted trees [9]. Any such algorithm could be used to produce an embedding. We chose a radial layout algorithm that positions the root node of the tree at the origin and positions its descendants in concentric circles.

The main idea of the layout algorithm is to define a sequence of consecutive disks, $D_1 \subset D_2 \subset D_3 \subset \cdots$, with radii $r_1 < r_2 < r_3 < \cdots$. Then we compute an angular wedge at each node such that the subtree rooted at that node is contain entirely within the angular wedge. The root node is positioned at the origin and the nodes of depth k are arranged on the boundary of the disk D_k. We

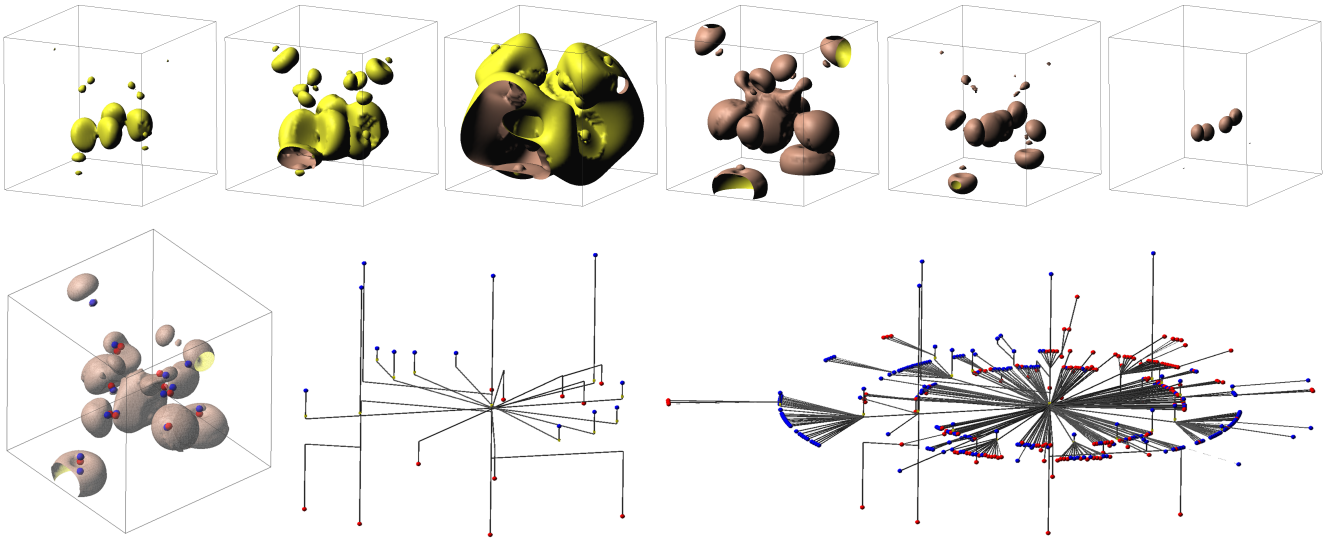


Figure 5: (top row) A sequence of level sets of the neghip dataset representing the spatial probability distribution of the electrons in a high potential protein molecule. (bottom-left) Critical points of the coarse scale topology, displayed together with a semi transparent level set. (bottom-middle) Contour Tree of the coarse scale topology. (bottom-right) Contour Tree of the full resolution topology.

require the ratio of consecutive radii to be a constant, $\rho = \frac{r_{k+1}}{r_k} > 1$. This guarantees the branches will be spread out nicely. If instead we fixed the difference between consecutive radii, then the ratio $\frac{r_{k+1}}{r_k} \to 1$ as $k \to \infty$, and the maximal size of the angular wedges goes to 0. Thus the subtrees of nodes far away from the origin will appear to be arranged along a straight line.

Figure 8(b) illustrates the algorithm for computing the angular wedge of a node N, which is on the boundary of the disk D_k. Let β be the angular wedge that has been computed for N. First, we can guarantee no self-intersections by ensuring that all arcs drawn from N to one of its children lie to the right of the tangent to the disk D_k at N. Otherwise, an arc could cross into the interior of the disk D_k, and may intersect an edge of the tree that has already been drawn. To ensure this is not the case we must restrict $\beta \leq 2cos^{-1}(\frac{r_k}{r_{k+1}}) = 2cos^{-1}(\frac{1}{\rho})$. In the figure we show the limiting case where $\beta = 2cos^{-1}(\frac{1}{\rho})$. In our implementation we use $\rho = \sqrt{2}$, thus we restrict $\beta < 2cos^{-1}(\frac{1}{\sqrt{2}}) = \frac{\pi}{2}$. However, one can see that it is only necessary to enforce this condition for the nodes on the boundary of the disk D_1, since we have chosen $\frac{r_{k+1}}{r_k}$ to be constant.

In figure 8 the children of the node N are the nodes N_1, N_2, and N_3. To compute the angular wedges β_i we partition the angle β proportionally to the sizes of the subtrees rooted at each node. If we let n_i be the number of leaves of the subtree rooted at N_i and n the number of leaves of the subtree rooted at N, then we have the following relations:

$$
\begin{aligned}
n &= n_1 + n_2 + n_3, \\
\beta &= \beta_1 + \beta_2 + \beta_3, \\
n_1 : n_2 : n_3 &= \beta_1 : \beta_2 : \beta_3.
\end{aligned}
$$

Therefore, we have that $\beta_i = \frac{n_i}{n}\beta \leq \beta$. Since $\beta < \frac{\pi}{2}$ then $\beta_i < \frac{\pi}{2}$ and we can guarantee that the subtree rooted at N_i is free of self-intersections.

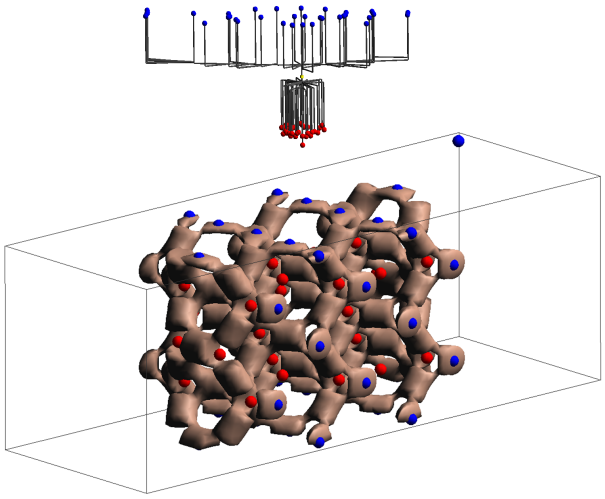
To compute the embedding of a hierarchical tree we use the parent-child relationship between branches to construct a rooted tree whose nodes are the branches of the hierarchical tree. Applying the layout algorithm above to this tree produces a planar embedding, which we use for the (x, y)-coordinates of nodes in the hierarchical tree. For each branch we assign these (x, y)-coordinates to all its interior nodes and its unpaired endpoint(s). As stated above the z-coordinate of each node is assigned the function value of the corresponding critical point in the scalar field. The branches are then visualized as "L" shapes, where the base of the "L" connects the branch to its parent along a horizontal line at the height of the paired endpoint.

Visualization In this section we demonstrate the use of the multi-resolution contour tree by showing how to produce output sensitive visualizations. We visualize the contour tree by embedding the nodes of the tree at the location of the critical points it space. Arcs are drawn as straight line segments connecting the endpoints. Thus a branch is drawn as a chain of connected arc segments. Since the branches are sorted by their persistence we always draw the branches with greater persistence first (see Figures 5 and 6).

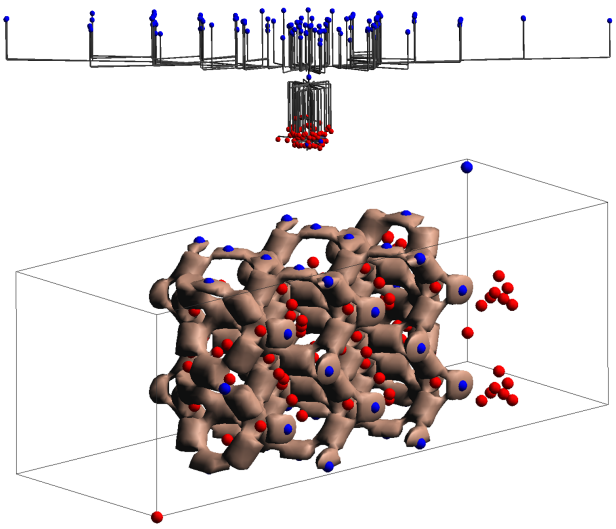
The branch decomposition representation of the CT allows for uniform or adaptive refinement of the tree. Uniform simplification is achieved by interrupting the drawing process when the first branch with persistence less than a specified value is reached. Adaptive simplification is almost as easy: before each branch is visualized it is tested to see if it satisfies the adaptive criterion. In Figure 10 the adaptive refinement is demonstrated using a spatial criterion. The criterion is to test if the bounding box of a branch intersects a user-provided bounding box describing a region of interest. The bounding box of a branch, B, is defined to be the bounding box of the region in space that contains all the critical points that must be canceled in order to simplify B.

The bounding boxes of each branch in the multi-resolution contour tree are computed by a simple recursive algorithm that merges the bounding box of the branch's endpoints with the bounding boxes of each of its child branches. After the bounding boxes have been computed the user is allowed to select a region of interest by manipulating a bounding box in the embedding space. We can now adaptively extract a CT that has a greater level of detail in the region of interest by only visualizing those branches whose bounding boxes intersect the user defined box.

Although this test is very simple it demonstrates a wide range of possibilities for user interaction with the CT. Adaptive refinement



(a) Coarse scale topology displayed on the abstract orrery interface (top) and in the original embedded space with a level set (bottom).



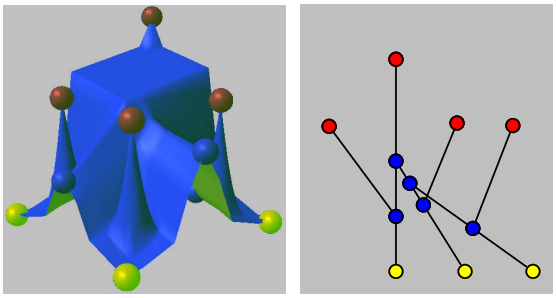
(b) Full resolution topology displayed on the abstract orrery interface (top) and in the original embedded space with a level set (bottom).

Figure 6: Topology of the electron density distribution in a simulation of a silicium grid. The topology can be presented incrementally from coarse (a) to fine (b).

becomes increasingly important when one begins to visualize large contour trees. In this case a view-dependent criterion would be useful in speeding up the rendering of the tree, as well as reducing the amount of information the user has to take it. Using such a criterion one would not render branches if they are outside the viewing area, too far away, or too small to see.

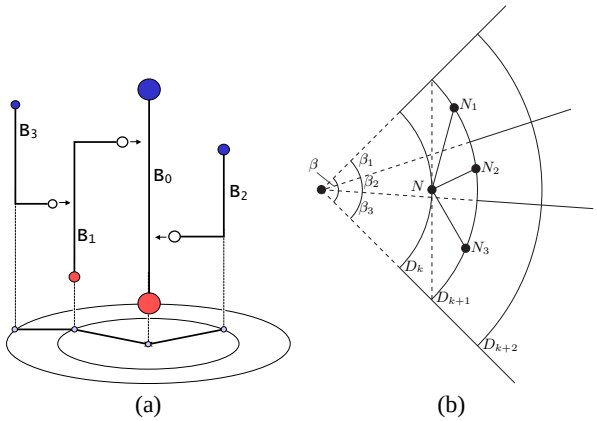
4 Conclusions

We have provided a data structure for representing multi-resolution Contour Trees and a simple algorithm for their construction. For piece-wise linear functions on simply-connected domains of any dimension the scheme can be proven to be topologically correct.



(a) (b)

Figure 7: (a) A simple terrain model. The Morse function f is the vertical elevation. The critical points are highlighted with spheres of different colors: four red maxima, three yellow minima and five blue saddles. (b) Standard 2D layout of the Contour Tree with the y coordinate equal to the function value of the critical points and the x coordinate used to minimize self-intersections. In this case there is always at least one self-intersection.



(a) (b)

Figure 8: (a) Orrery-like arrangement of the hierarchical decomposition of the Contour Tree in Figure 3. (b) Computation of the angular wedges β_1, β_2, and β_3 for the nodes N_1, N_2, and N_3 that are children of N. With this angular wedge distribution the layout has not self intersections.

The use of this data structure has been demonstrated by showing how to adaptively extract output sensitive contour trees.

We plan to extend the work on proving topological correctness for general volumetric domains. These datasets pose a problem because it is more difficult to determine cancellations for critical points that do not create junctions or bifurcations. In three-dimensional fields there can be two types of saddle points. This makes it possible for the two types of saddle points to form a pair than can be canceled, however, it is difficult to test if a pair of this type can be canceled.

References

[1] Chandrajit L. Bajaj, Valerio Pascucci, and Daniel R. Schikore. The contour spectrum. In Roni Yagel and Hans Hagen, editors, *IEEE Visualization ́97*, pages 167–175. IEEE, November 1997.

[2] Peer-Timo Bremer, Herbert Edelsbrunner, Bernd Hamann, and Valerio Pascucci. A multi-resolution data structure for two-dimensional Morse functions. In *Proceeding of IEEE Conference on Visualization*, pages 139–146, October 2003.

[3] H. Carr and J. Snoeyink. Path seeds and flexible isosurfaces - using topology for exploratory visualization. In *Proceeding of IEEE TCVG Symposium on Visualization (VisSym '03)*, pages 49–58, Grenoble, Fr, May 2003.

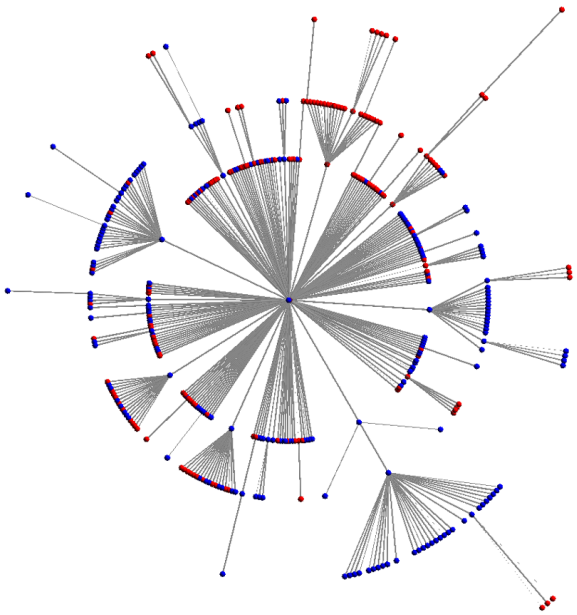


Figure 9: Projection to the xy plane of the orrery layout for the full resolution Contour Tree of the NegHip dataset shown on the bottom-right side of Figure 5

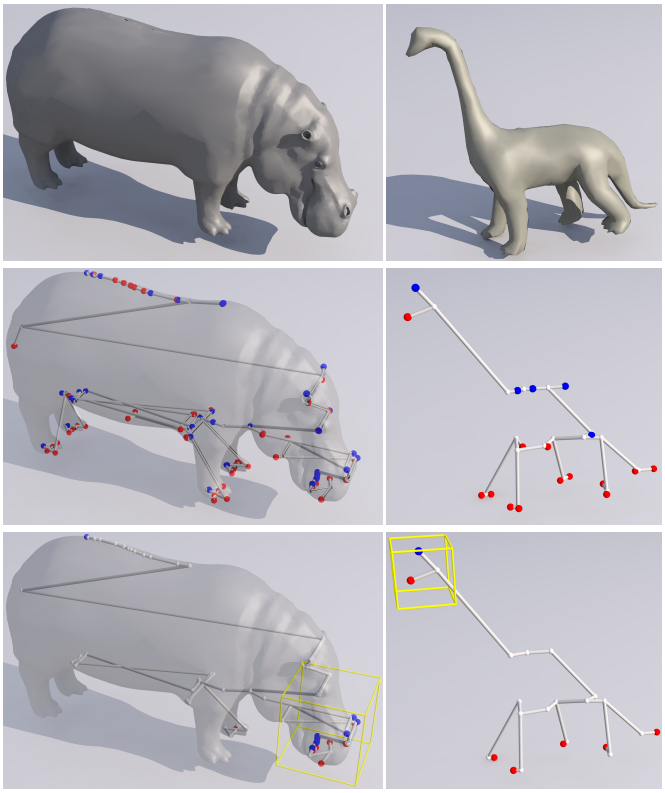


Figure 10: (top row) Triangulated models of a hippopotamus and dinosaur. (middle row) Complete Contour Trees drawn with its critical points in their original position. (bottom row) Adaptive refinement of the trees with full detail only around the head of the model.

[4] H. Carr, J. Snoeyink, and M. van de Panne. Simplifying flexible isosurfaces using local geometric measures. In *IEEE Visualization*, pages 497–504, October 2004.

[5] Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 918–926, January 2000.

[6] Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. *Computational Geometry Theory and Applications*, 2001. To Appear (extended abstract appeared at SODA 2000).

[7] Kree Cole-McLaughlin, Herbert Edelsbrunner, John Harer, Vijay Natarajan, and Valerio Pascucci. Loops in reeb graphs of 2-manifolds. In *ACM Symposium on Computational Geometry*, pages 344–350, July 2003.

[8] Giuseppe di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

[9] Giuseppe di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

[10] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. In *Proceeding of The 41st Annual Symposium on Foundations of Computer Science*. IEEE, November 2000.

[11] Herbert Edelsbrunner and Ernst P. Mucke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. In *Symposium on Computational Geometry*, pages 118–133, 1988.

[12] A. T. Fomenko and T. L. Kunii, editors. *Topological Modeling for Visualization*. Springer-Verlag, Tokyo, 1997.

[13] Thomas Gerstner and Renato Pajarola. Topology preserving and controlled topology simplifying multiresolution isosurface extraction. In T. Ertl, B. Hamann, and A. Varshney, editors, *Proceedings Visualization 2000*, pages 259–266, 2000.

[14] M. Hilaga, Y. Shinagawa, T. Komura, and T. L. Kunii. Topology matching for full automatic similarity estimation of 3d shapes. In *ACM SIGGRAPH*, pages 203–212, August 2001.

[15] E. Kleiberg, H. van de Wetering, and J.J. van Wijk. Botanical visualization of huge hierarchies. In *Proceedings IEEE Symposium on Information Visualization (InfoVis'2001)*, pages 87–94, 2001.

[16] Eleftherios Koutsofios and Stephen C. North. *Drawing graphs with dot*. Murray Hill, NJ.

[17] P. Lindstrom and V. Pascucci. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):239–254, July-September 2002.

[18] Valerio Pascucci and Kree Cole-McLaughlin. Efficient computation of the topology of the level sets. In *IEEE Visualization*, pages 187–194, October 2002.

[19] G. Reeb. Sur les points singuliers d'une forme de pfaff completement integrable ou d'une fonction numerique. *Comptes Rendus Acad. Sciences Paris*, 222:847–849, 1946.

[20] Y. Shinagawa, T. L. Kunii, H. Sato, and M. Ibusuki. Modeling the contact of two complex objects: With an application to characterizing dental articulations. *Computers and Graphics*, 19:21–28, 1995.

[21] Y. Shinagawa and T.L. Kunii. Constructing a Reeb graph automatically from cross sections. *IEEE Computer Graphics and Applications*, 11:44–51, November 1991.

[22] S. Takahashi, Y. Takeshima, and I. Fujishiro. Topological volume skeletonization and its application to transfer function design. *Graphical Models*, 66(1):24–49, 2004.

[23] Alexandru Telea and Jarke J. van Wijk. Simplified representation of vector fields. In *Proceedings of the conference on Visualization '99*, pages 35–42. IEEE Computer Society Press, 1999.

[24] Xavier Tricoche, Gerik Scheuermann, and Hans Hagen. A topology simplification method for 2d vector fields. In *Proceedings of the conference on Visualization '00*, pages 359–366. IEEE Computer Society Press, 2000.

[25] Edward R. Tufte. *Envisioning Information*. Graphics Press LLC, Cheshire, Connecticut, 1990.

[26] Gunther H. Weber and Gerik Scheuermann. *Automating Transfer Function Design Based on Topology Analysis*. 2004.