Simplifying the topology of volume datasets: an opportunistic approach

Andrzej Szymczak and James Vanderhyde
College of Computing
Georgia Tech
{andrzej, jamesv}@cc.gatech.edu

Understanding isosurfaces and contours (their connected components) is important for the analysis as well as effective visualization of 3D scalar fields. The topological changes that the contours undergo as the isovalue varies are typically represented using the contour tree, which can be obtained from the input scalar field by collapsing every contour to a single point. Contour trees are known to provide useful information, allowing one to find interesting isovalues and contours, speed up computations involving isosurfaces or contours, or analyze or visualize the scalar field's qualitative structure. However, the applicability of contour trees can, in many cases, be problematic because of their large size.

Morse theory relates the contour topology changes to critical points in the underlying scalar fields. We describe a simple algorithm that can decrease the number of critical points in a regularly sampled volume dataset. The procedure produces a perturbed version of the input volume that has fewer critical points but, at the same time, is guaranteed to be less than a user-specified threshold away from the input volume (in the supremum norm sense). Because the input and output volumes are close, the algorithm preserves the most stable topological features of the scalar field. Although we do not guarantee that the number of critical points in the output volume is minimum among all volumes within the threshold away from the input dataset, our experiments demonstrate that the procedure is quite effective for a variety of input data types. Apart from reducing the size of the contour tree, it also reduces the topological complexity of individual isosurfaces.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling; Curve, Surface, Solid, and Object Representations; I.3.6 [**Computer Graphics**]: Methodology and Techniques; Graphics Data Structures and Data Types

Additional Key Words and Phrases: Isosurface, Topology, Genus

1. INTRODUCTION

We propose a simple method allowing one to perturb a regularly sampled scalar field by an amount bounded by a prescribed threshold so that its topological complexity (the number of critical points) is reduced. As a result, both the size of the contour tree (which describes the topological changes that contours undergo as the isovalue is varied) and the topological complexity of individual isosurfaces decrease.

Algorithms such as the one proposed in this paper can be a useful part of the scalar field analysis pipeline, making the contour topology of large and complex datasets easier to comprehend and, in particular, opening new applications for contour trees as tools for understanding and visualizing the qualitative structure of scalar fields.

The approach we take in this paper is a direct extension of the isosurface topology simplification algorithm introduced in [Szymczak and Vanderhyde 2003], and it is based on a topology-preserving voxel removal as the fundamental primitive operation. The goal of [Szymczak and Vanderhyde 2003] is to simplify the topology of a single isosurface. In order to do that, we remove ('carve') the voxels in order of decreasing value (a dual variant would do the same in the opposite order). Assuming that the isovalue of interest is 0 and voxels in the volume bounded by the isosurface are negative, we remove only positive voxels and put a bound on the number of topology-altering voxel removals (which, in most practical cases, turns out to be equivalent to imposing a bound on the genus of the output isosurface). We also require that topology-altering removals are performed only if topology-preserving ones are not possible. After the procedure terminates, we change the values of all uncarved positive voxels to negative, obtaining a volume close to the input volume but with a topologically simplified isosurface.

In this paper, we eventually carve *all* voxels. We perform a topology-altering removal of a voxel of value I only if no voxels having value above $I - \tau$ can be removed while preserving the topology, where τ is a user-specified threshold. After the carving procedure terminates, we alter the voxel values to make them decrease with the carving order. The procedure proposed here does not necessarily lead to optimal results (i.e. leave the minimum number of critical points), but our experiments show that it performs quite well for a variety of input data types. We also discuss a multipass variant of our algorithm that decreases the number of voxels having different values in the input and output volumes.

2. RELATED WORK

In this section, we discuss the relation of our results to other topology simplification algorithms. We also give an overview of the work on scalar field topology that motivated this paper.

2.1 Topology simplification

Reduction of topological complexity has been an active research topic in recent years. Several algorithms, suitable for many different types of input data have been proposed. Below, we focus on the algorithms most related to ours.

An algorithm for removing handles from *3D surfaces* is described in [Guskov and Wood 2001]. It works by finding non-contractible loops in the surface, cutting the surface along these loops and triangulating the resulting boundary loops. An algorithm based on the same principle but tailored to work with *isosurfaces* in volume data has been proposed in [Wood et al. 2002]. The paper [Szymczak and Vanderhyde 2003] achieves the same goal by first computing the signed distance transform for the isosurface and then executing the carving procedure outlined in Section 1. An essentially identical algorithm developed for binary volume data had been introduced in [Aktouf et al. 2002]. Similar ideas had been used to preserve the topology of level sets in [Han et al. 2001; 2002].

Topology is the main focus of the work discussed above. However, there is a large body of work on combining geometry and topology simplification. For CAD models, a method of simplifying both geometrically and topologically has been proposed in [El-Sana and Varshney 1997]. In the volumetric domain, low pass filtering can be seen as a simple way of removing noise from the data, including the topological noise. However, it may cause undesirable loss of detail and it does not provide control over the amount of perturbation. Methods allowing control or preservation of the isosurface topology as the underlying scalar field is simplified are described in [Gerstner and Pajarola 2000; Chiang and Lu 2003a].

2.2 Descriptions of topology of scalar fields

The work described in this paper has been mainly motivated by the need for *simple and comprehensible* descriptions of the topology of scalar fields that are needed for large and complex data. Scalar field topology is most commonly described using the *contour tree*, a special case of the Reeb graph [Reeb 1946]. Contour trees have received a significant amount of interest in recent years. Efficient algorithms for computing contour trees are described in [Carr et al. 2003; Chiang and Lu 2003b; Tarasov and Vyalyi 1998; Pascucci and Cole-Mclaughlin 2002]. The method allowing to label contour tree edges with the Betti numbers and the numbers of boundary loops of the corresponding contours is described in [Pascucci and Cole-Mclaughlin 2002] and [Berglund and A.Szymczak 2004]. Contour trees have found applications in scalar field visualization, where they can help accelerate isosurface extraction [Bajaj et al. 1998], find contours with user-defined properties [Bajaj et al. 1997] or provide a framework for interactive exploration of the scalar fields [Carr et al. 2004; Carr 2004].

Unfortunately, typical volume data (especially obtained using a physical acquisition technique) may have a very large number of critical points. The flexible isosurface technique [Carr et al. 2004] allows to simplify the contour tree by applying elementary operations like leaf pruning or vertex removal until it becomes more comprehensive and therefore useful for interactive exploration of contours. The goal of this paper is similar, but we focus on computing a perturbation of the *scalar field*, within a user-specified threshold, that would reduce its number of critical points. By superimposing the threshold on the perturbation, we ensure that the persistent (in the sense equivalent to [Edelsbrunner et al. 2002; Edelsbrunner et al. 2001; Bremner et al. 2003]) topological features are present in the output volume. Although our procedure is not *guaranteed* to remove all features of persistence lower than the threshold, we have successfully used it to substantially reduce the topological complexity of a number of test datasets (Section 6). At the same time, it is very easy to implement. We believe it can represent a practical (albeit, at this point, only single-resolution) alternative to the Morse-Smale complex described in [Edelsbrunner et al. 2003], which is commonly regarded as challenging to implement in practice. Moreover, our method seems to be relatively easy to extend to higher dimensions. We are planning to report on the performance of such an extension elsewhere.

3. PRELIMINARY DEFINITIONS

Before we describe our algorithm in detail, we need to introduce the necessary terminology. The input dataset is a regularly sampled volume of resolution $n_x \times n_y \times n_z$, in which the spacing between the samples in the x, y and z dimensions is equal to d_x, d_y and d_z (respectively). By a voxel we shall mean an axis-oriented parallelepiped centered at one of the samples and with edge lengths of d_x, d_y and d_z in dimensions x, y and z. This means that there are $n_x n_y n_z$ voxels and they are in one-to-one correspondence with the samples (their centers). Therefore, we shall think of each voxel as having a value (equal to the value at its corresponding sample).

Let $\mathcal{S}_0$ be the set of all voxels. Below we shall consider subsets $\mathcal{S} \subset \mathcal{S}_0$. Each subset $\mathcal{S}$ defines:

(a). A solid in $\mathcal{R}^3$, the union of all voxels in $\mathcal{S}$. This solid will be denoted by $|\mathcal{S}|$.

(b). A surface in $\mathcal{R}^3$, the isosurface corresponding to the isovalue of 0, computed using the algorithm of [Bhaniramka et al. 2000] assuming that the values at the voxels in $\mathcal{S}$ are all equal to -1 and the values of all voxels in $\mathcal{S}_0 \setminus \mathcal{S}$ are equal to 1. This isosurface is denoted by $\langle \mathcal{S} \rangle$.

Note that the *topology* of $\langle \mathcal{S} \rangle$ is the same as the topology of the isosurface computed using the algorithm of [Bhaniramka et al. 2000] assuming that the values of all voxels in $\mathcal{S}$ are arbitrary but lower than the isovalue and the values of all voxels outside $\mathcal{S}$ are arbitrary but higher than the isovalue. Specific values of voxels do influence the geometry of the isosurface, but they do not alter its topology. Throughout this paper, all isosurfaces we discuss are computed using the algorithm of [Bhaniramka et al. 2000].

4. THE SINGLE-PASS ALGORITHM

Our algorithm takes a regularly sampled volume dataset and a user-specified persistence threshold τ as its input. It produces a volume dataset of the same resolution, in which value of every voxel V is in $[I - \tau, I]$, where I is the value of V in the original volume. If desired, by adding $\tau/2$ to the value of each voxel one can obtain a volume in which the voxel values are within $\tau/2$ from the values of the corresponding voxels in the input volume. The output volume will, in practice, have less critical points and simpler isosurfaces than the input volume.

The procedure is a sweep algorithm, in which voxels are processed in order of decreasing value. For each voxel we consider its removal (carving) from the current voxel set $\mathcal{S}$ (initially, all voxels). Such a removal may cause $\langle \mathcal{S} \rangle$ to change the topology. Assuming that every voxel is actually removed, voxels for which a topology change occurs are the critical voxels. Our procedure works by perturbing the carving order of voxels: keeping it close to ordering by value, but at the same time attempting to decrease the number of topology-altering voxel removals.

Persistence, as introduced in [Edelsbrunner et al. 2001], measures the 'life span' of topological features resulting from such topology changes. For example, if a topologically nontrivial cycle is created as a result of removal of a voxel V and, later on, it disappears as a result of removal of another voxel W, then the life span of that cycle can be defined as the difference of the values of W and V.

Algebraic topological tools (homology theory) make this definition fully precise [Edelsbrunner et al. 2001], allowing to express the creation and disappearance of cycles in purely algebraic terms. In 2D, persistence essentially trivializes: it reduces to describing how connected components of the set $\mathcal{S}$ and its complement split or join as the voxels are being removed [Edelsbrunner et al. 2002; Bremner et al. 2003]. Algorithmically, this is done by employing the union-find datastructure, in a way similar to the split and join tree algorithm of [Carr et al. 2003]. However, in higher dimensions no comparably simple and computationally efficient (at least in the worst case) approach is known. Morse-Smale complexes [Edelsbrunner et al. 2003] are a direct generalization of the framework that was implemented with success in 2D in [Edelsbrunner et al. 2002; Bremner et al. 2003], but they are much more complex in 3D and so far have not been implemented in practice.

Our algorithm attempts to find a way around this complexity by making simple opportunistic decisions. Whenever we are considering a voxel V for removal but we detect that it would cause a topological change, we delay the removal, hoping to find a different removal order that would remove V without the need for any topology altering voxel removals. Clearly, such a delay should be bounded by the user-specified bound on the error τ. More precisely, if the values of two voxels in the input volume differ by more than τ, the one with the larger value must be removed first. This idea motivates the carving procedure described below.

4.1 Carving

We use a priority queue of voxels in order to determine their processing order. Initially, the value of a voxel is used as its priority and the priority queue contains all voxels. The set $\mathcal{S}$, subject to the carving operations, is initialized to hold all voxels, i.e. to $\mathcal{S}_0$.

We iteratively take a voxel V off the queue. If that voxel does not belong to $\mathcal{S}$, it is ignored and we proceed to the next voxel. Otherwise, we test whether the removal of V from $\mathcal{S}$ would preserve or change the topology of $\langle \mathcal{S} \rangle$. (This test is described in detail in Section 4.2.)

In the first case, V is removed from $\mathcal{S}$. We also update the priority queue by inserting into it all voxels for which the removal of V might have changed the outcome of the topology preservation test. The test depends on the status of all voxels in the 26-neighborhood of V (i.e. whether they had been carved before or not). Therefore, we need to insert all neighbors of V that belong to $\mathcal{S}$ into the queue. The values of the inserted voxels are used as their priorities. Note that the same priority queue update is performed each time a voxel is removed from $\mathcal{S}$.

In the second case (i.e. when removing V from $\mathcal{S}$ would change the topology of $\langle \mathcal{S} \rangle$), what we do depends on whether the priority of the voxel is equal to its value I or not. If it is, we push the same voxel back onto the queue with priority $I - \tau$. Intuitively, this means delaying its removal in hope of finding a way to remove it without topological change later on. Setting the priority to $I - \tau$ ensures that the maximum delay until V is actually removed is bounded by τ.

If the priority of V is different from I, V is a delayed voxel. Its priority has to be equal to $I - \tau$. In this case, we remove the voxel even though this might change the topology of $\langle \mathcal{S} \rangle$ and update the priority queue as described earlier. The voxel V will be critical in the output volume if its removal alters the topology of $\langle \mathcal{S} \rangle$.

4.2 Testing for topology change

The test for topology preservation of $\langle \mathcal{S} \rangle$ under the removal of a voxel V that we use in the carving procedure is a simple extension of the tests used in [Aktouf et al. 2002; Han et al. 2001; 2002; Szymczak and Vanderhyde 2003].

If V is an interior voxel of the domain, i.e. $V \subset \text{int } |\mathcal{S}_0|$, it is in fact exactly the same. Let $I(V, \mathcal{S})$ denote the intersection of the boundary of V (denoted by δV) and $|\mathcal{S} \setminus \{V\}|$. Removal of V from $\mathcal{S}$ does not change the topology of $\langle \mathcal{S} \rangle$ if and only if both $I(V, \mathcal{S})$ and its complement in δV, i.e. $\delta V \setminus I(V, S)$ have exactly one connected component.

If V is a boundary voxel, the test described above is not sufficient. For example, if $\mathcal{S} = \mathcal{S}_0$, then any boundary voxel would pass the topology preservation test described above (provided $n_x, n_y, n_z \geq 2$). However this is not correct: removing such a voxel would create a component in $\langle \mathcal{S} \rangle$, and therefore change its topology. Let $B(V) = \delta V \cap \delta |\mathcal{S}_0|$ be the intersection of the voxel's boundary and the boundary of the domain. It is not hard to see that the removal of V from $\mathcal{S}$ preserves the topology of $\langle \mathcal{S} \rangle$ if and only if each of the two sets: $I(V, \mathcal{S})$ and $I'(V, \mathcal{S}) = I(V, \mathcal{S}) \cup B(V)$ as well as their complements in δV all have exactly one connected component.

Clearly, these tests can be completed in constant time. In practice, one can use a binary lookup table to speed up the running time of the test. We implement the table as follows. For a voxel, we consider all subsets of the set of voxels in its 26-neighborhood. Such subsets are in one-to-one correspondence with binary sequences of length 26 and therefore can be indexed by integers in the range $0 \ldots 2^{26} - 1$. Consider the intersection of the boundary of the voxel and the union of all voxels in the subset corresponding to an integer $i \in \{0 \ldots 2^{26} - 1\}$. We set the i-th entry of the lookup table to 1 if and only if both that intersection and its complement relative to the voxel's boundary each have one connected component. With the lookup table described above, one can decide whether removal of a voxel V from S preserves the topology as follows. Consider the following two subsets of the the set of all neighbors of V:

(i). the set of all neighbors of V that are in $\mathcal{S}$

(ii). the set of all neighbors of V that are either in $\mathcal{S}$ or outside the domain.

The removal of V from $\mathcal{S}$ does not change the topology of $\langle \mathcal{S} \rangle$ if and only if the entries of the lookup table corresponding to both of the above sets are equal to 1.

The lookup table described above requires 2^{26} bits or 8MB of memory. Clearly, it is possible to reduce the number of configurations and therefore also the size of the lookup table by exploiting symmetry, but implementing this does not appear to be worthwhile in practice, at least in the 3D case. It is interesting to note that in the 4D case, the full lookup table would require 2^{80} bits and therefore seems to be unpractical at the time of this writing. A simple implementation of the 4D variant of the topology preservation test can be based on the connected component count and the Euler characteristics using the idea of [Delfinado and Edelsbrunner 1995].

4.3 Updating the voxel values

After the carving procedure terminates (i.e. all voxels are removed from $\mathcal{S}$), we alter the voxel values to make their ordering by decreasing value identical to the

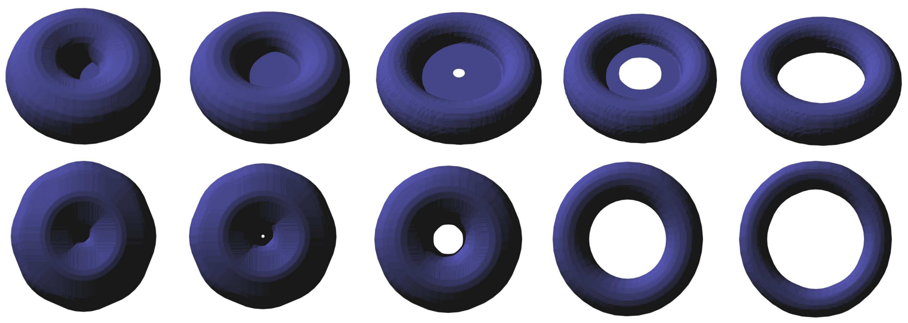

Fig. 1. Snapshots of the carving process for the single-pass algorithm (top row) and the second pass of the multipass algorithm (bottom row) for the signed distance function for the torus. Both procedures require the same number of topology changes (only one of them is shown in the figure). The carving order produced by the multipass algorithm much better reflects the ordering of voxels by value. The new values of voxels computed using the algorithm described in Section 4.3 are also closer to the original values of voxels for the multipass algorithm than for the single-pass algorithm. Notice that the single pass algorithm assigns the same (only infinitesimally different) values to all voxels on the 'membrane' that initially closes the through-hole in the torus but is eventually decided to be pinched because of high persistence of the saddle in its center.

carving order. In order to do that, we scan voxels in the order they were carved, keeping track of the minimum value of a voxel seen so far (including the current one). For every voxel, we set its new value to this minimum value. Clearly, that minimum value does not have to strictly decrease as we move from voxel to voxel and therefore degeneracies (like two neighboring voxels with the same value) might result. We use the carving order to resolve such degeneracies: whenever values of two voxels in the output volume are compared and they turn out to be equal, we declare the value at the one that has been carved first larger. This is equivalent to simulating a perturbation of values by an infinitesimal quantity that is strictly decreasing with respect to the carving order.

Notice that the value of a voxel in the output volume cannot be larger than the value of the same voxel in the input volume. Moreover, every voxel having value I is removed before any voxel with value less than $I - \tau$. This proves the following statement.

Observation 1. *If the value of a voxel in the input volume is I then the value of the same voxel in the output volume has to be in $[I - \tau, I]$.*

5. ITERATIVE IMPROVEMENT OF THE CARVING ORDER

The single pass algorithm already achieves the goals laid out for this paper: it significantly decreases the number of critical points (Section 6) and keeps the maximum error between the input and output volume within the user specified threshold. However, the carving order computed by the single pass algorithm is clearly suboptimal, as shown in Figure 1. In this section we describe a procedure that can improve the carving order, making it closer to the ordering of voxels by value. This improved order will also cause a smaller number of voxel values to be changed in the update phase (Section 4.3).

5.1 Multipass algorithm: underlying idea

The first pass of the algorithm is identical to the single-pass version of Section 4. Each pass aims to improve the carving order of the previous pass, while attempting to preserve the structure and order of its topological events. The overall structure of each of the passes is the same. We use a priority queue of voxels as in the single pass variant. Initially, all voxels are inserted into the queue with priority equal to their value *in the input volume* (updating the values described in Section 4.3 is performed after the last pass) and we set $\mathcal{S} := \mathcal{S}_0$. Let $\mathcal{C}$ be the set of all critical voxels in the volume produced in the preceding pass (recall that such voxels are defined by the carving order used in that pass and can be determined as all voxels whose removal alters the topology of the isosurface $\langle \mathcal{S} \rangle$). In a loop, we extract a voxel V from the queue and consider it for removal from $\mathcal{S}$. The removals are governed by the following rules:

1. The voxels in $\mathcal{C}$ have to be removed in the same order as in the preceding pass. Note that this order is the same as their ordering by value in the input volume.

2. If a voxel W was removed after a voxel $V \in \mathcal{C}$ in the preceding pass, W has to be removed after V also in the current pass. Basically, this means that, right after (and right before) the removal of a voxel $V \in \mathcal{C}$, the set of uncarved voxels in the current pass is a superset of the set of uncarved voxels in the preceding pass right after (before) the removal of V.

3. If the value of a voxel V in the original volume exceeds the value of a voxel W in the original volume by τ or more, V has to be removed before W in the current pass. This is done to ensure that Observation 1 holds after updating the voxel values based on the traversal order used in any (in particular, the final) pass.

4. Unless prevented by the rules **1.**, **2.** or **3.**, a voxel whose removal does not change the topology of $\langle \mathcal{S} \rangle$ or a voxel in $\mathcal{C}$ whose set of neighbors in $\mathcal{S}$ is the same as in the previous pass at the moment of its removal is decided to be removed as soon as it is considered for removal. In particular, this means that the topology changes induced by the removal V in the current and previous passes are equivalent. Our early implementation actually compared the topological changes introduced by the two removals, but this turned out to offer little advantage over simply comparing the neighborhoods.

Note that, in general, it is possible that removals of some voxels that do not belong to $\mathcal{C}$ alter the topology of $\langle \mathcal{S} \rangle$ (which means that new critical voxels are introduced in passes other than the first one). However, we found out that it is very uncommon: in fact, new critical points are rarely found even in the second pass. Our current implementation always terminates after two passes. The second and later passes (if used) are described in more detail below.

5.2 Algorithm statement

Before we start a new pass, we compute $\mathcal{C}$, the set of critical voxels relative to the carving order from the preceding pass. Let the voxels in $\mathcal{C}$ (in order of being carved in the preceding pass) be $V_1, V_2, \dots V_k$. Let $\mathcal{N}_i$ be the set of all voxels in the 26-neighborhood of V_i that were carved after V_i. Let $\mathcal{V}_i$ be the set of all voxels

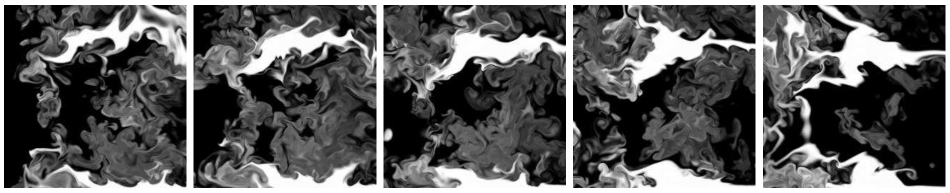

Fig. 2. Slices through the fluid simulation dataset.

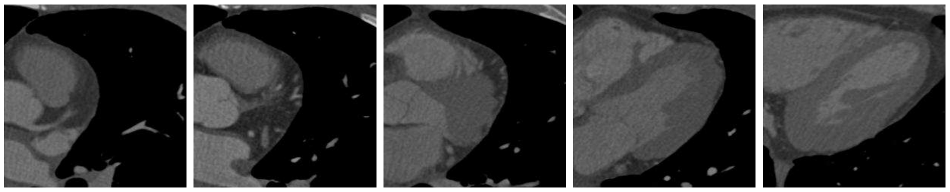

Fig. 3. Slices through the CT scan dataset.

carved no later than V_i but (if $i > 1$) after V_{i-1} for $i \in \{1, 2, \ldots, k\}$. Clearly, the sets $\mathcal{V}_i$ form a decomposition of the set of all voxels $\mathcal{S}_0$ into disjoint subsets.

We initialize the set $\mathcal{S}$ (the set of voxels not carved yet) to the set of all voxels $\mathcal{S}_0$. We use a counter i to keep track of the next voxel in $\mathcal{C}$ to be carved (initially, 1). We insert all voxels in $\mathcal{V}_1$ into the queue, using their values as priorities. Until the priority queue becomes empty, we extract a voxel V from the queue. Now, the action depends on that voxel:

(i). If V does not belong to $\mathcal{S}$, we ignore it and proceed to the next voxel.

(ii). If the removal of V does not change the topology of $\mathcal{S}$, we remove V, update the queue (i.e. insert all neighbors of V belonging to $\mathcal{S}$ which were removed before V_i in the preceding pass into the queue; values of the inserted voxels are used as their priorities). Then we proceed to the next voxel.

(iii). If $V = V_i \in \mathcal{C}$ and the intersection of all voxels in the 26-neighborhood of V and $\mathcal{S}$ is equal to $\mathcal{N}_i$ (note that the first voxel falls into this category), we remove V from $\mathcal{S}$, increment i and push all voxels in $\mathcal{S}_i$ onto the queue (using their values as priorities). Then we proceed to the next voxel.

(iv). If the priority of V is equal to its value I and the conditions in **(ii)** and **(iii)** fail, we push it back onto the queue with priority $I - \tau$. Then we proceed to the next voxel.

(v). If the priority of V is different from its value (in which case it has to be equal to $I - \tau$), we remove V from $\mathcal{S}$ and update the queue as in **(ii)**. Additionally, if $V \in \mathcal{C}$, we increment i and push all voxels in $\mathcal{S}_i$ onto the queue (using their values as priorities). Then we proceed to the next voxel.

6. RESULTS

We have tested our algorithm for a number of volume datasets. Below, we focus our attention on three representative datasets: an approximate signed distance function from the well-known the Buddha model [Curless and Levoy 1996] sampled

Threshold	original	1	3	5	10	15	20	30
Critical Points	23832	209	95	71	43	25	17	3
CT nodes	12531	82	40	30	16	10	6	2
β_0 and β_1	7, 22	1, 14	1, 12	1, 12	1, 10	1, 6	1, 4	1, 0
Altered voxels (1 iteration)	N/A	20244	40809	65750	152173	241968	353392	417176
Altered voxels (after 2 iterations)	N/A	16477	22030	24310	98853	134575	195835	417135

Table I. Results for the signed distance volume for the Buddha model. We used 0 as the isovalue for isosurfaces whose Betti numbers are listed in the table. The isosurfaces were always closed (therefore $\beta_2 = \beta_0$).

Threshold	original	0.5	2.5	4.5	8.5	16.5	32.5	64.5	128.5	256.5
Critical points	321483	198467	82818	47671	21873	7450	1988	482	56	2
CT nodes	130425	63706	20718	10796	4608	1814	574	152	24	2
β_0	507	507	362	260	182	124	60	17	2	1
β_1	2786	2786	2450	2079	1500	811	180	38	2	0
β_2	402	402	269	178	111	67	24	5	0	0
Altered voxels (1 pass)	N/A	0	241597	611270	695387	809054	950320	1069321	1134626	1129825
Altered voxels (2 passes)	N/A	0	232670	608357	693583	800258	931539	1037439	1127575	1129825

Table II. Results for fluid simulation dataset (Betti numbers shown correspond to isovalue 127.5). Note that the intensities of all voxels are integers and therefore only infinitesimal perturbation is performed for $\tau = 0.5$ – this is why the reported count of altered voxels is zero.

Threshold	original	0.05	0.1	0.2	0.6	1.0	1.4	2.0
Critical points	584879	46346	7780	1395	166	92	53	2
CT nodes	274338	24953	4713	853	65	32	16	2
β_0	1081	417	245	82	14	7	6	1
β_1	3339	997	381	109	25	2	1	0
β_2	886	307	165	41	2	1	0	0
Altered voxels (1 pass)	N/A	1172476	1432351	1513028	3171618	3878776	4481524	4487685
Altered voxels (2 passes)	N/A	1157489	1408416	1561595	3148997	3861051	4450089	4487685

Table III. Results for the CT scan; Betti numbers of isosurface for isovalue of 0.7 are shown.

on a $181 \times 181 \times 434$ grid (it has positive values inside the Buddha and negative values outside), a $256 \times 256 \times 128$ time slice of a fluid dynamics simulation and a $241 \times 281 \times 191$ subvolume of a chest CT scan. On an 850MHz Pentium III machine, our implementation of the two-pass algorithm requires up to about 6 minutes to complete for each of the three datasets. The datasets vary substantially in character: the first two are synthetic, the last one is acquired. Synthetic datasets are known to be easier for topological algorithms as they contain less topological noise [Carr 2004]. The signed distance function is relatively simple while the fluid simulation dataset is quite complex (Figure 2). The CT scan volume, like most acquired datasets, is noisy: it contains a high number of low persistence critical points. Slices through the CT scan volume are shown in Figure 3.

First, we compare several performance characteristics of our algorithm for each of the three datasets and for a variety of persistence thresholds:

(a). The number of critical points in the output dataset

(b). The number of contour tree nodes in the output dataset (after removing all regular nodes, i.e. nodes having one neighbor above and one neighbor below)

(c). The Betti numbers of the isosurface corresponding to an arbitrarily selected isovalue

(d). The number of voxels in the output volume that have a different value than in the input volume (we think of it as a measure of the amount of change applied to the data); we compare these numbers for the one-pass and two-pass version of

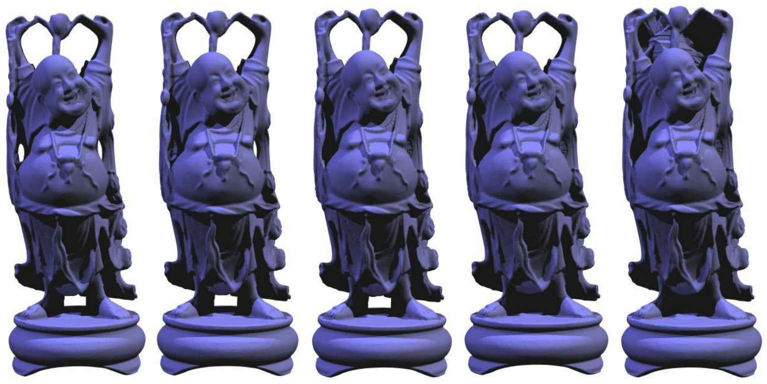

Fig. 4. Isosurfaces corresponding to the isovalue of zero for thresholds 1, 10, 15, 20 and 30.

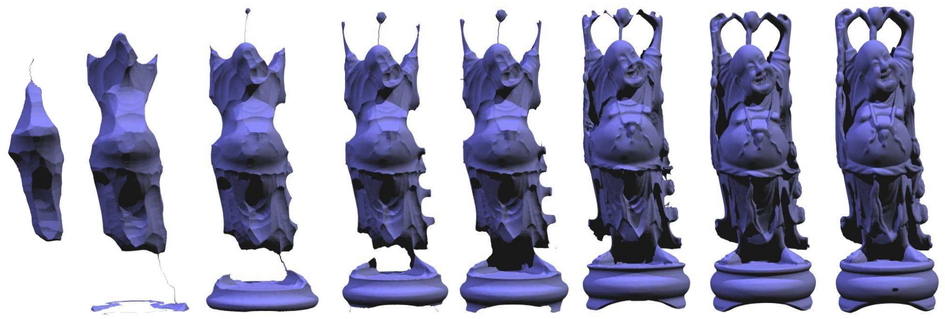

Fig. 5. Snapshots of the carving procedure for $\tau = 20$. Note that all surfaces throughout the process are connected and have genus between 0 and 2.

our algorithm. When comparing the voxel values, we disregard the infinitesimal perturbation.

The results for each of the three models are shown in Tables I, II and III. The tables show that our procedure can highly reduce the number of critical points in a volume dataset and the size of its contour tree. In particular, for all datasets the maximum simplification has a small number of critical points (2 or 3). This is true for most datasets we experimented with, except for those described in Section 6.1. The tables also demonstrate that the isosurfaces in the output volume are also topologically simplified. Finally, the signed distance datasets demonstrate the usefulness of the second pass: the two-pass version of the algorithm alters fewer voxel values. The difference is much smaller for all other, less regular, datasets.

Isosurfaces corresponding to isovalue of 0 for different simplified versions of the signed distance dataset are shown in Figure 4. Snapshots of the carving process (or, equivalently, isosurfaces in the output volume corresponding to different isovalues) are shown in Figure 5. Contour trees for the topologically simplified CT scan (rendered using graphviz [Gansner et al. 1993; Gansner and North 2000]) are shown in Figure 6.

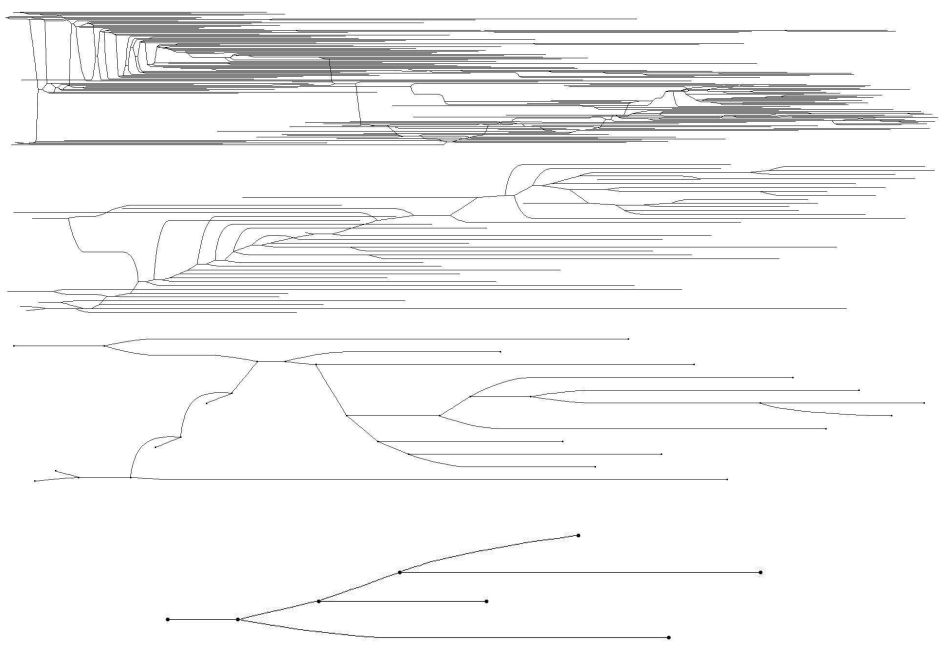

Fig. 6. Contour trees for the topologically simplified CT scan (thresholds: 0.3, 0.5, 1.0, 1.6).

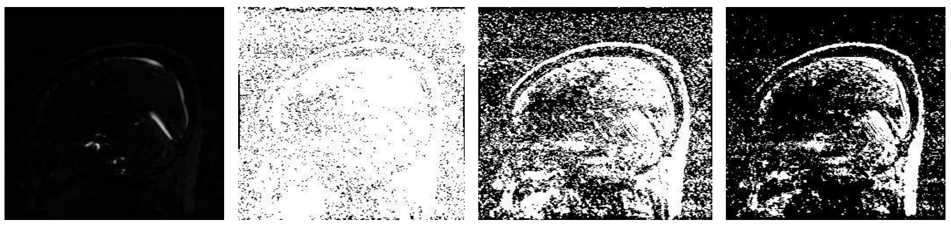

Fig. 7. A section through the MRA dataset (left) and images obtained from it by thresholding using small thresholds to show the noise.

6.1 Hard cases

We found out that the behavior described above is typical for relatively well-behaved datasets. However, we noticed that data containing flat and noisy regions may lead to worse results, in particular causing our algorithm to generate a large number of critical points even for very large thresholds (larger than the dynamic range of the input data). An example slice through such dataset (an MRA scan of the head) is shown in Figure 7. The resolution of this particular dataset is $256 \times 256 \times 60$. Our algorithm produced 37 critical points and 10 nodes in the simplified contour tree even for thresholds larger than the dynamic range of the dataset.

In the hard cases such as the one discussed above, we recommend removing the noise before using the algorithm described in this paper to simplify the topology.

Even very crude denoising like simply applying the box filter greatly improves the performance of our algorithm. In fact, for all realistic datasets we tested it on it enables reduction of the number of critical points to under 4 for large thresholds.

6.2 Quality of output: discussion and experimental evaluation

Our algorithm is not optimal: the number (and total complexity) of critical points in the output volume may not be minimum over all volumes within the threshold from the input volume. Below, we discuss a number of examples leading to suboptimal output. Then, we discuss experimental comparison of the topological complexity of the volume produced by our algorithm to a theoretical lower bound.

6.2.1 *Examples illustrating suboptimality.* Our procedure preserves the location of the global maximum of the input volume. If the global maximum is not on the boundary, it generates a volume with at least 3 critical points for any threshold: one corresponding to the global maximum, one corresponding to the point where the contour (or one of the contours if there are more than one) hits the boundary as the isovalue is decreased and one at the global minimum. On the other hand, if perturbation by more than the dynamic range of the dataset is allowed, it is possible to perturb the input volume to obtain one with only two critical points.

A more consequential example can be built based on one of the topological spaces that are contractible but not collapsible, like the House of two rooms [Steen and Seebach 1995]. The idea is similar to the one used in [Szymczak and Vanderhyde 2003]. Take the House of two rooms with walls only one voxel thick, and with no voxels that can be removed from it while preserving the topology. It can be obtained from the cube (union of all voxels) by means of topology-preserving carving operations. Assign values to voxels so that they decrease with the carving order. Let m be the lowest of these values. Assign an arbitrary value less than m to every voxel of the House. It is not hard to see that, regardless of the threshold, our procedure will be carving the House. At the moment only the voxels in the House walls are left, no one can be removed without a topology change and a critical point will be generated. By embedding several Houses in a volume as described above we can build volumes for which our procedure would generate an arbitrarily high number of critical points for any threshold.

Yet another example leading to suboptimal output is shown in Figure 8. If the set $\mathcal{S}$ at some stage of carving is as shown on the left of the Figure, there are at least two substantially different ways for the carving to proceed until $\mathcal{S}$ consists of only red voxels:

a. Perform a topology-altering removal of a grey voxel (other than the one that remains on the right of Figure 8), followed by six topology-preserving removals of grey voxels, leading to the set $\mathcal{S}$ shown on the right of the Figure. Then, carve all voxels other than the red ones. This requires two additional topology-altering operations.

b. Remove one of the yellow voxels (altering the topology) and then all voxels other than red ones (this is possible without any additional topology-altering voxel removals).

Notice that the first carving order leads to 3 critical points, while the second carving

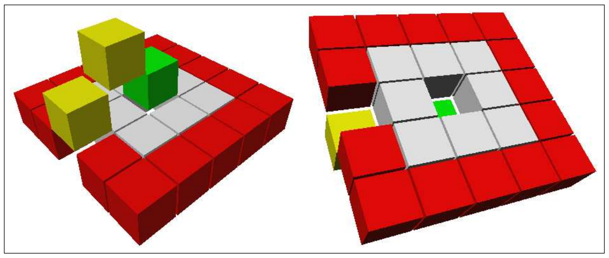
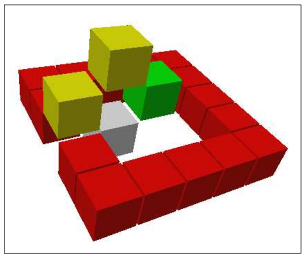

Fig. 8. An example showing a possibility of suboptimal output. Left: two views of the set $\mathcal{S}$ at some moment of the carving procedure. None of the voxels shown is on the boundary of the dataset. The values of all the red voxels are 0. The values of all other voxels are $\tau + 1$, so that they all have to be removed before any of the red voxels. Right: a set obtained from the set shown on the left by means of one topology-altering removal of a grey voxel followed by 6 topology-preserving grey voxel removals.

order generates only 1 critical point. By slightly changing the values of the non-red voxels one can, in fact, force the first carving order and therefore suboptimal output of our algorithm.

6.2.2 *Experimental comparison to theoretical lower bound.* In order to asses the quality of output of the opportunistic approach to topology simplification, we set up a simple experiment. We produced a small $32 \times 32 \times 16$ dataset by subsampling the fluid simulation dataset used in Section 6. Then, we applied the box filter to obtain its smoother versions. The intensities of voxels in all the test volumes were integers in $[0, 255]$. In order to be able to use the available homology computation tools as well as the theoretical results on persistence, we used a variation of the algorithm, which, instead of postponing carving voxels which would change the topology of $\langle \mathcal{S} \rangle$, postpones carving voxels which would alter the topology of $|\mathcal{S}|$. One could think of this procedure as an opportunistic way to minimize the number of topological changes to the volume *inside* an isosurface (or the set of points with value below the isovalue – frequently called the *sublevel set*) that happen as the isovalue is varied from the maximum to the minimum value over the entire dataset.

Our main motivation is to compare the total complexity of topology changes in the simplified volume (with threshold τ) to the total number of persistent homology generators in the input volume. To measure the total complexity of the topology changes that sublevel sets undergo we use the following quantity:

$$\kappa_\tau = \sum_{V \in \mathcal{S}_0} \text{dim coker } H_*(i_V) \tag{1}$$

where the summation extends over all voxels V and i_V is the inclusion map of the union $L_V^<$ of all voxels having value strictly lower than V in the simplified volume into the union $L_V^{\leq}$ of all such voxels and V. Note that $L_V^<$ (respectively, $L_V^{\leq}$) could be equivalently defined as the union of all voxels that remained in $\mathcal{S}$ right after (respectively, right before) V was carved during simplification. In all computations, we use homology with coefficients taken from the field of real numbers. Notice that the summation can be restricted to all critical voxels without changing the result (if i_V is a homotopy equivalence, the corresponding summand of Equation 1 is

Box filter iterations	τ	8	16	32	64
4	κ_τ	6,8,2	5,4,0	2,2,0	1,1,0
	ξ_τ	6,8,2	5,4,0	2,2,0	1,1,0
1	κ_τ	23,103,47	9,44,14	1,9,2	1,2,0
	ξ_τ	22,95,47	9,43,12	1,8,2	1,2,0
0	κ_τ	58,468,266	25,240,151	6,87,49	1,13,6
	ξ_τ	57,334,243	21,160,140	6,63,42	1,10,5

Table IV. Comparison of κ_τ and ξ_τ (in dimensions zero, one and two) for different values of the persistence threshold τ. The table on the right has been obtained for subsampled fluid simulation dataset and the table on the left for its smoothed version.

vanishes). κ_τ measures the total number of homology generators of $|\mathcal{S}|$ 'destroyed' during the carving procedure: each summand of Equation 1 counts the number of linearly independent generators in $L_V^{\leq}$ that cannot be represented as cycles in $L_V^{<}$.

We compare κ_τ to the number of all persistent homology generators in the input dataset. The persistent generators are computed as the following sum:

$$\xi_\tau = \sum_{n\in\{0,1,\ldots,255\}} \text{rank } H_*(j_{n+1,n+1+\tau}) - \text{rank } H_*(j_{n,n+1+\tau}), \tag{2}$$

where $j_{p,q}$ is the inclusion map of the union of voxels with value less or equal to p (denoted by S_p) into the union of voxels having value less or equal to q (i.e. S_q). Each summand of (2) contributes the codimension of im $H_*(j_{n,n+1+\tau})$ relative to im $H_*(j_{n+1,n+1+\tau})$, or the number of generators that need to be added to the set of generators of im $H_*(j_{n,n+1+\tau})$ in order to obtain a set of generators for im $H_*(j_{n+1,n+1+\tau})$. Notice that the added generators are persistent (they belong to $H_*(S_{n+1+\tau})$). Moreover, they can be represented as cycles in $H_*(S_{n+1})$ and are created when the isovalue increases over $n+1$, since they cannot be represented as cycles in $H_*(S_n)$. In particular, they correspond to persistent cycles in the sense of [Edelsbrunner et al. 2002]. We conclude that Equation (2) expresses the total number of all cycles of persistence τ or more.

It is a simple exercise in homology theory to show that $\xi_\tau \leq \kappa_\tau$. The performance of our algorithm can be evaluated by examining the difference between κ_τ and ξ_τ. In the case of our test dataset, the results are shown in Table IV. All the homology computations were performed using the CHomP package [Pilarczyk 2004; Kaczynski et al. 2004; Mischaikow et al. 2004]. For the smoothest version of the test dataset (4 iterations of the box filter), κ_τ and ξ_τ were the same for all tested thresholds, which means that our procedure performed optimally. For the one of intermediate smoothness, differences between κ_τ and ξ_τ appeared, but they remained rather small. Finally, for the noisiest, unfiltered test dataset, the differences were much larger, although the total κ_τ (sum of κ_τ over homology of all dimensions) still generally remained within 30 per cent from the ξ_τ.

7. CONCLUSION AND FUTURE WORK

We have presented a simple-to-implement procedure allowing computation of a perturbation of a volume dataset having significantly fewer critical points. Several interesting variations of our algorithm could possibly be of interest for specific applications. For example, instead of attempting to remove the voxels so that introduction of critical points is avoided, one might want to avoid only critical points of certain types, as in the experiment described in Section 6.2. One could make the threshold τ depend on a sample, its intensity or the intensity distribution of its neighborhood, rather than keeping it constant throughout the entire dataset. In time-dependent data, it might be of interest to avoid topology changes of contours as they evolve in time.

Clearly, topologically simpler volumes may be easier to use in concrete applications: since they have fewer contours, more precise and computationally expensive shape descriptors may be used to find the ones with desired characteristics. Smaller datastructures are needed to keep isosurface seeds (like path seeds described in [Carr 2004]). Simplified contour trees become easier to visualize and compare. In particular, it would be interesting to apply tree matching algorithms to find correspondences between critical points in different volumes; this is a fundamental problem in numerous applications.

Improvements of the procedure described in this paper could incorporate smoothness measures into the voxel priorities, attempting to reduce the sensitivity of the procedure to noise mentioned in Section 6.1. Finally, it would be interesting to perform a rigorous comparison to the techniques based on Morse Complexes as well as a rigorous analysis of how effective our procedure is in removing low persistent topological features.

REFERENCES

Aktouf, Z., Bertrand, G., and Perroton, L. 2002. A three-dimensional holes closing algorithm. *Pattern Recognition Letters*, 523–531.

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

Bajaj, C., Pascucci, V., and Schikore, D. 1997. The contour spectrum. In *Proc. IEEE Visualization 1997*. 167–175.

Berglund, N. and A.Szymczak. 2004. Making contour trees subdomain-aware. In *Proceedings of the 16th Canadian Conference on Computational Geometry (CCCG'04)*. 188–191.

Bhaniramka, P., Wenger, R., and Crawfis, R. 2000. Iso-contouring in higher dimensions. In *IEEE Visualization 2000*. 267–273.

Bremner, P.-T., Edelsbrunner, H., Hamann, B., and Pascucci, V. 2003. A multi-resolution data structure for two-dimensional mores-smale functions. In *Proc. IEEE Visualization 2003*. 139–146.

Carr, H. 2004. Topological manipulation of isosurfaces. Ph.D. thesis, University of British Columbia.

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

Carr, H., Snoeyink, J., and van de Panne, M. 2004. Simplifying flexible isosurfaces using local geometric measures. In *Proc. IEEE Visualization 2004*.

Chiang, Y. and Lu, X. 2003a. Progressive simplification of tetrahedral meshes preserving all isosurface topologies.

Chiang, Y.-J. and Lu, X. 2003b. Simple and optimal output-sensitive computation of contour trees. Tech. Rep. TR-CIS-2003-02, Polytechnic University. June.

Curless, B. and Levoy, M. 1996. A volumetric method for building complex models from range images. In *Proc. of SIGGRAPH 1996*. 4–9.

Delfinado, C. J. A. and Edelsbrunner, H. 1995. An incremental algorithm for betti numbers of simplicial complexes on the 3-sphere. *Computer Aided Geometric Design 12,* 7, 771–784.

EDELSBRUNNER, H., HARER, H., AND ZOMORODIAN, A. 2001. Hierarchical morse-smale complexes for piecewise linear 2-manifolds. In *Symp. on Computational Geometry*. ACM, ACM Press, New York, NY, USA, 70–79.

EDELSBRUNNER, H., HARER, J., NATARAJAN, V., AND PASCUCCI, V. 2003. Morse-smale complexes for piecewise linear 3-manifolds. In *Proc. 19th Ann. Sympos. Comput. Geom.* 361–370.

EDELSBRUNNER, H., LETSCHER, D., AND ZOMORODIAN, A. 2002. Topological persistence and simplification. *Discrete Comput. Geom. 28*, 511–533.

EL-SANA, J. AND VARSHNEY, A. 1997. Controlled simplification of genus for polygonal models. In *Proc. IEEE Visualization '97*, R. Yagel and H. Hagen, Eds. 403–412.

GANSNER, E. R., KOUTSOFIOS, E., NORTH, S. C., AND VO, K.-P. 1993. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering 19,* 3, 214–230.

GANSNER, E. R. AND NORTH, S. C. 2000. An open graph visualization system and its applications to software engineering. *Software-Practice and Experience 30,* 11, 1203–1233.

GERSTNER, T. AND PAJAROLA, R. 2000. Topology preserving and controlled topology simplifying multiresolution isosurface extraction. In *Proc. IEEE Visualization 2000*, T. Ertl, B. Hamann, and A. Varshney, Eds. 259–266.

GUSKOV, I. AND WOOD, Z. 2001. Topological noise removal. In *Proc. Graphics Interface 2001*, B. Watson and J. W. Buchanan, Eds. 19–26.

HAN, X., XU, C., AND PRINCE, J. L. 2001. A topology preserving deformable model using level sets. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR2001)*. 765–770.

HAN, X., XU, C., AND PRINCE, J. L. 2002. A topology preserving geometric deformable model and its application in brain cortical surface reconstruction. In *Geometric Level Set Methods in Imaging, Vision and Graphics*, S. Osher and N. Paragios, Eds. Springer Verlag.

KACZYNSKI, T., MISCHAIKOW, K., AND MROZEK, M. 2004. *Computational Homology.* Applied Mathematical Sciences, vol. 157. Springer-Verlag.

MISCHAIKOW, K., MROZEK, M., AND PILARCZYK, P. 2004. Graph approach to the computation of the homology of continuous maps. *Foundations of Computational Mathematics, to appear*.

PASCUCCI, V. AND COLE-MCLAUGHLIN, K. 2002. Efficient computation of the topology of level sets. In *Proc. IEEE Visualization 2002*. 187–194.

PILARCZYK, P. 2004. Homology software. `http://www.math.gatech.edu/~ chom/`.

REEB, G. 1946. Sur les points singuliers d une forme de pfaff compl'ement integrable ou d une fonction numerique. *Comptes Rendus de L Academie ses Seances, Paris 222*, 847–849.

STEEN, L. A. AND SEEBACH, J. A. 1995. *Counterexamples in Topology.* Dover Publications.

SZYMCZAK, A. AND VANDERHYDE, J. 2003. Extraction of topologically simple isosurfaces from volume datasets. In *Proc. IEEE Visualization 2003*. 67–74.

TARASOV, S. P. AND VYALYI, M. N. 1998. Construction of contour trees in 3d in $o(n \log n)$ steps. In *Proc. 14th Ann. Sympos. Comput. Geom.*

WOOD, Z., HOPPE, H., DESBRUN, M., AND SCHRÖDER, P. 2002. Isosurface topology simplification. Tech. Rep. MSR-TR-2002-28, Microsoft Research.