The Topological Effects of Smoothing

Sohail Shafii, Scott E. Dillard, Mario Hlawitschka, and Bernd Hamann

Abstract—Scientific data sets generated by numerical simulations or experimental measurements often contain a substantial amount of noise. Smoothing the data removes noise but can have potentially drastic effects on the qualitative nature of the data, thereby influencing its characterization and visualization via topological analysis, for example. We propose a method to track topological changes throughout the smoothing process. As a pre-processing step, we over-smooth the data and collect a list of topological events, specifically the creation and destruction of extremal points. During rendering, it is possible to select the number of topological events by interactively manipulating a merging parameter. The result that a specific amount of smoothing has on the topology of the data is illustrated using a topology-derived transfer function that relates region connectivity of the smoothed data to the original regions of the unsmoothed data. This approach enables visual as well as quantitative analysis of the topological effects of smoothing.

Index Terms—Volume visualization, visualization techniques and methodologies, smoothing.

1 Introduction

Topological analysis is an increasingly important approach for visualization and analysis of scientific data in many fields, with relevance for data sets originating in a broad spectrum of applications, ranging from materials science to medicine. Transfer function design for volume rendering can be viewed as an application of topological characterization, where certain regions of data sets corresponding to ranges of function values are assigned optical properties such as color and opacity. These regions allow a user to easily identify and delineate significant structures in a data set. Transfer function methods based on simple thresholding remain attractive due to their simplicity, reproducibility, and minimal user involvement, but they cannot be used when significant noise is present in the data. In addition, noise generally leads to isosurfaces that contain many spurious surface components, which complicates queries involving the number, size, and distribution of different regions. When having to process and visualize noisy data, smoothing is always an alluring option, leading to isosurfaces with fewer components and smoother boundaries. However, smoothing has the potential to drastically alter the data to the point where one draws invalid conclusions.

We present a two-stage method to track and visualize the topological changes that occur when smoothing a scalar-valued data set defined over a volumetric, three-dimensional (3D) domain. During the tracking stage, we apply smoothing up to a point where a meaningful interpretation of the data is no longer possible. We track the topological changes that occur, specifically the creation and destruction of extremal points that merge with other topological features that persist. In the visualization stage that follows, the user can select the topological events that affect the original noisy data set by applying labels (or colors) to different portions of the data. To summarize, we have developed a general topological analysis method that requires little input from the user.

We first provide an introduction to Morse theory and contour trees in Section 2. We describe the methods used in Section 3 and discuss the results obtained with our methods in Section 4. Future research possibilities are covered in Section 5.

- *Scott E. Dillard (Scott.Dillard@pnl.gov) is with the Pacific Northwest National Laboratory, Richland, WA.*
- *Sohail Shafii (ssshafii@ucdavis.edu), Mario Hlawitschka (hlawitschka@ucdavis.edu), and Bernd Hamann (hamann@cs.ucdavis.edu) are with the Institute for Data Analysis and Visualization (IDAV), Department of Computer Science, University of California, Davis, CA.*

2 Background

This research builds upon methods developed in the last decade that apply Morse theory to computer graphics and visualization. Morse theory [1] studies the topology of manifolds by way of level sets of scalar functions defined over those manifolds. A *level set* of a scalar function $f : X \to \mathbb{R}$ is the inverse image of a value y, $f^{-1}(y)$, which is the set of points $x \in X$ such that $f(x) = y$. A *contour* of a level set is one connected component of that set. As y is varied, contours merge and split at *critical points*, which are those points $x \in X$ where $\nabla f(x) = 0$. Additionally, contours can change their surface topology by adding or removing handles. For instance, in the three-dimensional domain of a scalar-valued trivariate function, a spherical contour can change into a torus. Critical points can be divided into two types: *extrema*, which are points where f attains a local maximum or minimum, and *saddles*, which are points where the topology of the level set changes by either the number of connected components within a level set or the topology of a single connected component of that level set.

In our approach, we employ the *contour tree*, an accepted means for describing the level set topology of scalar functions. A special case of the contour tree is known as the *Reeb graph* [2], which is obtained from a function by collapsing each contour of each level set to a point while maintaining the connectivity of contours between level sets. Formally, it is the quotient space of $X/\sim$ where the equivalence relation $\sim$ is defined such that $a \sim b$ when a and b are both contained in the same contour of the same level set. If X is simply connected (i.e., having no handles) the Reeb graph contains

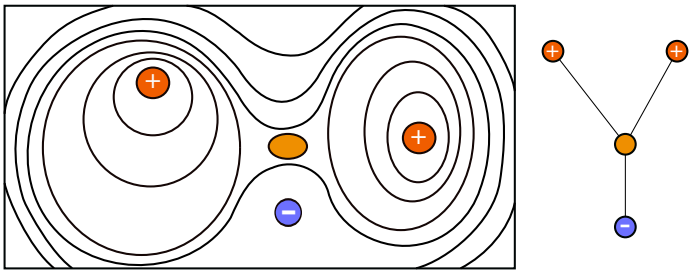


Fig. 1. A topographic map of two mountains on the left and the corresponding contour tree on the right, where each contour line is collapsed to a point on the tree's arcs. The red points are maxima ($+$ symbol), the orange point is the saddle (the oval between the mountains), and the blue point is the minimum ($-$ symbol). For simplicity, extrema on the boundaries are ignored. As one moves up from the bottom of the contour tree towards increasing isovalue or height, he/she can imagine moving from a single component, the minimum, up to the saddle. At the saddle, the component splits into two mountains, which end at two separate maxima.

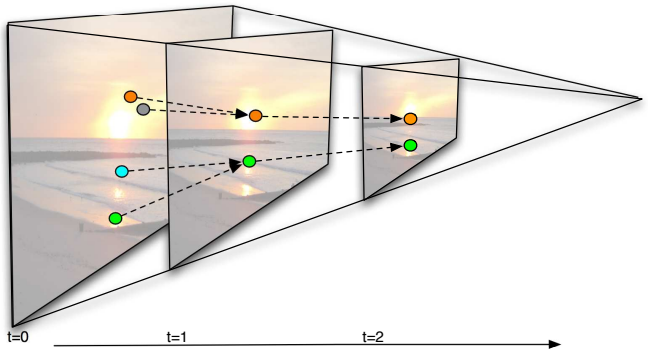


Fig. 2. A visual representation of the scale space as a pyramid that indicates the loss of information when the image is smoothed. The variable "t" indicates time or, in our case, the number of times an image is smoothed. As one moves forward in time, one can can observe the movement of extrema by stacking the images produced by repeated smoothing on top of each other. In this particular example, the colored nodes represent extrema and the dashed lines indicate paths. From time step 0 to 2, the initial four extrema in the image merge into two extrema after the other extrema were smoothed out.

no cycles and becomes a contour tree [3]. See Figure 1 for an example. This distinction between a contour tree and Reeb graph is important because there are algorithms with efficient worst-case running time for computing contour trees [4], [5], [6] that take advantage of this acyclic property, and simply connected domains are common in imaging applications.

Some early work in the application of Morse theory to computer graphics and visualization was performed by Shinagawa *et al.* [8] who did some early work on reconstructing 3-D surfaces, van Kreveld *et al.* [9] who recognized that the Reeb graph can be used to accelerate isosurface extraction by identifying "seed cells" from which to begin surface construction, and Bajaj *et al.* [10] who proposed the *contour spectrum* to plot geometric as well as topological properties of isosurfaces against their isovalues. Later, Carr *et al.* [11] developed the idea of the contour tree as a metaphor for direct manipulation of isosurfaces for interactive exploration, used in their implementation via an interface supporting flexible isosurface extraction. Takahashi *et al.* [12] used the contour tree for automatic transfer function design, and Weber *et al.* [13] extended the flexible isosurface idea for volume rendering. In parallel, research has been carried out exploring the application of the Morse-Smale complex, a topological structure that is similar to the Reeb graph but contains more information, to problems in computer graphics and visualization. For more detail, we refer to the papers by Bremer *et al.* [14], Gyulassay *et al.* [15], [16], and Laney *et al.* [17].

There is an independent line of research in the computer vision community which combines Morse theory with scale-space analysis. Scale-space methods seek to characterize the structure of an image by observing its evolution under repeated convolution with a Gaussian filter kernel. The scale-space image is the $(n+1)$-dimensional image formed by stacking together each n-dimensional image that is the result of Gaussian smoothing with the previous image in the stack. A visual illustration is shown in Figure 2. Lifshitz and Pizer [18] proposed to track the extrema in each image in this stack as they trace out paths through the additional scale dimension. Kuijper and Florack [19], [20] explored this idea further and defined a segmentation by level sets that contain scale-space saddles, which are the saddles of the $(n+1)$-dimensional function. These points correspond to the birth[1] or death of extrema during the smoothing process. Gingold and Zorin [21] proposed and analyzed an algorithm that controls the topological events while an image is being modified by Laplacian, sharpening, and anisotropic diffusion filters.

There also exist methods that analyze time-varying contours or Morse functions. Szymczak [22] introduced a method to describe the cumulative changes of contours in time-varying data sets. Edelsbrunner *et al.* [23] developed an algorithm that constructs a time-varying Reeb graph by tracking the motion of critical points along so-called *Jacobi curves* [24]. The birth and death of critical points in time occur at degenerate critical points along these curve-like sets. If time is viewed as the additional scale-space dimension, then these degenerate critical points are also the scale-space saddles. Sohn and Bajaj [25] also proposed a method to track topology by computing correspondence information between contour trees from one time step to the next and applied their method to visualization. An area of correspondence is caused by overlapping regions related to contours between a contour tree in time step t and another contour tree from time step $t+1$. Silver and Wang [26] introduced an algorithm that tracks volume features in unstructured data sets by detecting features

1. It is counter-intuitive that blurring based on a Gaussian operator creates new extrema, but this indeed can happen as Lifshitz and Pizer [18] pointed out.

that overlap each other in time-varying data sets. In a similar paper, Chen *et al* [27] discuss a feature extraction and tracking algorithm that works with time-varying data sets while dealing with multiple resolutions and processors. Fujishiro *et al.* [28] introduced an interface which visualized the changes in topology (over time) through a pixel map called *index space*. Keller and Bertram [29] created a tool that visualized surfaces defined by time-varying Reeb Graphs which are modified by operations on nodes and edges. Takahasi *et al.* [30] and Nieda *et al.* [31] explored controlling topological transitions in the context of shape morphing.

Unlike previous time-varying methods based on Morse theory, we do not explicitly track and store a topological data structure. We also do not compute areas of overlap as discussed in [26]. Instead, we track the appearance, disappearance, and movement of extrema in an image under the influence of continuous smoothing and use the tracking information to label the contour tree of the original (unsmoothed) image. Thus, the benefit of our method is due to the fact that each tracking stage from time step to time step is simple to implement, since we are not explicitly computing correspondence information between contour trees as shown in [25].

Tracking extrema discretely over time is similar to tracking the evolution of blobs in scale-space theory, which relate to features in an image and cover an area larger than individual extrema, as discussed by Lindeberg [32]. Unlike tracking and linking blobs from one time step to the next, extrema tracking is more difficult because extrema move faster than large features. Nevertheless, both methods allow one to simplify an image and label its essential structure separate from noisy elements.

The labeling procedure we employ effectively simplifies a noisy contour tree, which is similar to other topological simplification methods. Carr *et al.* [33] illustrate how one can simplify a contour tree by "reducing" vertices without changing the essential structure of the tree. Takahasi *et al.* [34], [35] remove candidate links from the contour tree that consist of nodes that are most likely related to noise. Weber *et al.* [13] modify the scalar field in real-time using graphics hardware to correspond to a simplified contour tree. Unlike our method, existing simplification techniques influence the structure of the contour tree. Our contour tree's structure remains untouched while the labels of the nodes are modified in an effort to create less noisy regions.

When seeking to implement a topological method robustly, one must carefully consider how the concepts of Morse theory, which considers smooth functions, are used in an actual implementation that can only represent discrete values. The most common representation of a function is a set of discrete sample points which are extended to the rest of the domain by interpolation. The interpolation method depends on the application, but two very common methods are linear interpolation for triangulated domains and trilinear interpolation for meshes consisting of rectilinear cuboids as mesh building blocks. When possible, another option is to decompose a given three-dimensional grid of samples into tetrahedra, which makes many algorithms simpler but has been shown to introduce artifacts [36].

We have based our work on the piecewise-trilinear interpolant for three-dimensional images. The same interpolant can be applied to both isosurfacing and volume rendering. Linear interpolation over tetrahedral elements can be used for volume rendering of rectilinear grid data but has significant speed and artifact-related drawbacks. Topologies defined by modified marching cubes use case tables, or other so-called "digital" topologies, and cannot be applied to any form of volume rendering. Nielson [37] has described the topology of the trillinear interpolant to make robust algorithms possible. Notably, critical points of the piecewise-trilinear function can occur at sample points, a single saddle may occur within a face shared by cubic cells, and a pair of saddles may occur in the interior of a cubic cell. Nielson [37] described the analytical conditions for these points to occur, while Pascucci and Cole-McLaughlin [6] characterized face and cell saddles combinatorially by the gradient direction along edges between sample points. This combinatorial characterization is important for robust algorithms which are free from degenerate saddles and we make use of it.

While a precise combinatorial description has been given for the topological changes that occur within a time-varying linearly-interpolated function defined over tetrahedra [38], the same result for time-varying trilinear interpolation is still an open problem, although results in that direction have recently been given by Carr and Max [39]. We do not address the general-case problem of describing the evolution of trilinear critical points through time, rather we restrict our attention to the more tractable problem of tracking only extrema of the trilinear function.

3 METHODS

Our visualization pipeline is divided into two parts: tracking and rendering. In the tracking stage, we track extrema in a data set while applying a smoothing operator. In the rendering stage, the tracking results are used to simplify the contour tree. The data we consider are scalar values associated with the vertices of a hexahedral mesh, with trilinear interpolation used to extend these values into the faces and cells of this mesh. The application of Morse theory to discrete functions requires us to use a technique known as *symbolic perturbation*. Symbolic perturbation ensures that critical points are isolated: the piecewise-defined function attains a maximum or minimum only at vertices of the hexahedral mesh. To achieve this goal, every edge between two vertices sharing the same value is "tilted" so that one vertex' value is greater than the other. In practice, the position of a vertex in the image is used as a tie-breaker when comparing vertices with equal value.

The partial ordering of adjacent vertices can be represented by a directed edge based on the gradient, oriented to point toward the vertex with greater value. Maxima and minima can be recognized as those vertices that are sources or sinks in this discrete vector field, respectively. As noted by Pascucci and Cole-McLaughlin [6], the existence of saddle points is also determined by the configuration of these "gradient arrows" around a face or a cell, although their exact position and value

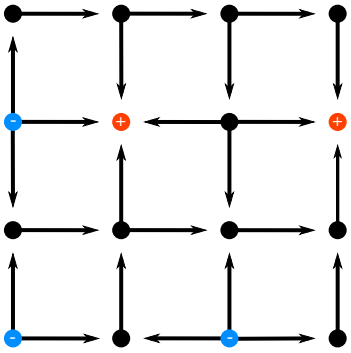
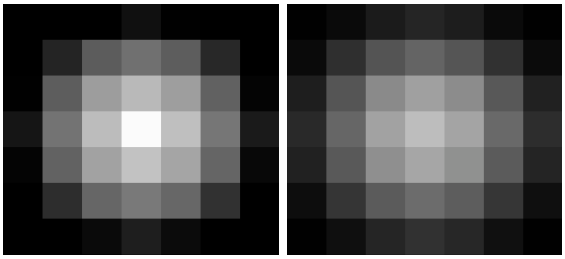


Fig. 3. Each data element in the grid (indicated by circles) has a scalar value. In addition, we store the discrete gradient vector field, i.e., an oriented graph using the edges of the grid. Maxima are red circles with a plus symbol; minima are blue symbols with a minus symbol.



(a) Gray-scale rendering of original function values.

(b) The effect of applying a 7×7 discrete Gaussian smoothing filter to gray-scale image.

Fig. 4. An example of a gray-scale image that is smoothed by a discrete 7 x 7 discrete Gaussian smoothing filter. Black pixels represent minimal function value and white pixels represent maximal function value, with varying grey tones indicating in-between values. During smoothing, each data element is assigned a value that is the weighted average of all points in the kernel centered at that data element.

must be computed numerically [37]. Figure 3 illustrates this discrete gradient field.

3.1 Smoothing and Tracking Extrema

The tracker analyzes each data element to observe movement of extrema. During tracking, we apply a Gaussian smoothing filter [40] to an image multiple times while following the movement of extrema for each data element independently.

3.1.1 Gaussian smoothing:

In two dimensions, the filter weights are computed using the two-dimensional Gaussian

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \qquad (1)$$

where σ is the standard deviation and σ^2 the variance. The fraction is a constant for a given variance and acts as a normalization factor. As smoothing with a Gaussian filter can be seen an analytical solution to a diffusion equation, σ^2 can be seen as the variance $E[(\xi-\mu)^2]$, where $\xi \sim N(\mu,\sigma)$ is a random variable with normal distribution (aka. Gaussian distribution). The effect of applying the filter multiple times to obtain discrete time steps is an iterative approach to solve this diffusion equation, which can be written analytically as a combined filter, G_n at time step $n > 0$ and can be computed by multiple convolutions of the Gaussian $G(\cdot,\cdot,\sigma)$, where

$$G_n(x,y,\sigma) = \frac{1}{2\pi n\sigma^2} e^{-\frac{x^2+y^2}{2n\sigma^2}} = G(x,y,\sigma_n) \qquad (2)$$

with $\sigma_n = n\sigma$. Although an analytical solution would be possible, we need all discrete time steps to perform the tracking. Therefore, doing multiple convolutions with a small filter is more efficient than increasing the filter size to compute the image for time step n analytically.

Lifshitz and Pizer [18] provided a detailed analysis of the Guassian filter and we refer the reader to that work for details regarding this elementary image processing operator.

Boundary conditions: As the filter is defined as a convolution of the image with the filter kernel, a proper definition of boundary conditions is important. Several standard methods used in image processing, namely *zero boundary condition*, *periodic boundary condition*, *cyclic boundary condition*, and *closest point boundary condition*, which can be implemented by extending the data set (padding) by at most half of the filter size at all boundaries. Another way of handling the boundary is to modify the filter at the boundary. Whereas some of the above approaches can be seen as a modified filter, it is possible to use the Gaussian weights inside the data set, but normalizing the data points at the end that the picture's "energy" is maintained, i.e., the integral over the filter inside the data domain remains one (or in terms of the diffusion operator, no particle/energy leaves the domain). For our tests, we chose the closest point condition, i.e., we implicitly pad the data set by copying the boundary values to the additional layers, because this approach seems to have the smallest influence on the boundary topology. This is especially important for the small test cases shown in this paper but does not necessarily influence the larger data sets shown in the Results section. In the larger data sets shown in this paper, the data domain is usually larger than the actual data and the boundary artifacts have a negligible influence on the important data. A detailed study of those artifacts is not part of this paper.

Discretization: We use a discrete version of this general Gaussian filter that uses only the function values in a finite voxel neighborhood around a voxel $[i,j,k]$ whose value we replace by a weighted combination of all the values in the chosen neighborhood. In our case, we use a three-dimensional discrete Gaussian filter that determines a weighted average of $7 \times 7 \times 7$ values replacing the values at the center voxel. The specific weights used for the voxels in this neighborhood are calculated using Equation 1; however, we exclude the constant and normalize by dividing each weight by the sum of all the weights, which is required to maintain the total "energy" of the data. Figure 4 illustrates a two-dimensional finite voxel neighborhood that is smoothed by a discrete Gaussian filter.

3.1.2 Fundamentals of Tracking:

Before tracking, we record the initial positions of the extrema in the original data set and label them. The smoothing filter is applied to produce a new image. As the new image values are copied back to the old image one value at a time from the first vertex to the last vertex, we track the behavior of each extremum. This smoothing and tracking process can be repeated to a specified number of iterations or *time steps*.

Because we modify only a single value at a time, determining the creation, destruction and movement of extrema is done for each vertex. We model the change in a vertex' value as a continuous motion. Since the existence of critical points is determined by the orientation of the discrete gradient arrows incident on that vertex, we only need to consider what happens as the orientation of these arrows changes. We decompose the movement of the vertex value into even smaller steps so that we flip exactly one gradient arrow at a time. Whenever a gradient arrow flips, we track the topological changes that occur.

These topological changes include cases when an extremum is created, when it is destroyed, and when it moves to an adjacent vertex. Any extremum that is created is "spurious" in the sense that it is an artifact of smoothing. To relate a new extremum to the rest of the critical points, we search in the vicinity of the location of its creation for another pre-existing extremum of the same type after the gradient arrow flipping occurred. For example, after a new maximum is created we walk along edges of the hexahedral mesh with gradient arrows that point "uphill" (in the direction of maximal gradient) until we reach another maximum. The newly created maximum is assigned a label that corresponds to this neighboring maximum. New minima are paired with an existing minima by walking along the direction opposite to the maximal gradient. By using this approach, an extremum of the original unsmoothed image can propagate its label to multiple extrema. We refer to these occurrences as *anti-cancelation* events, which relate to the case where spurious bumps that appear on the side of the mountain after smoothing would merge with the neighboring extrema (the mountain). Note that we relate new extrema with pre-existing extrema since we must use our tracking results to modify our static contour tree during rendering, which is based on the unsmoothed data set.

Extrema that disappear can also merge with labels of pre-existing extrema. For instance, when a gradient arrow flips an extremum might disappear without moving to a neighboring vertex. In this case we perform a similar search, walking uphill in the case of a maximum or downhill in the case of a minimum, until a neighboring extremum is found. One intuitive example that relates to disappearing extrema is the case of a small bump on the side of a mountain. As the entire mountain is smoothed, the small bump would disappear and one would walk uphill to merge with the entire mountain since that is the only surviving feature. *Cancelation* events such as this are recorded by noting the label of the extremum that was lost and that of the neighboring extremum.

In the cancelation and anti-cancelation cases, the neighboring extremum is known as the *parent*. Anti-cancelations duplicate the labels of one extrema while a cancelation occurs after all such labels have been removed due to smoothing. Since information from cancelation events enables us to find out how noisy features merge with other features, we use these events to influence the rendering stage.

We do not explicitly maintain the contour tree of the varying data set during the smoothing process, as we do not have a precise characterization of the behavior of saddles of a trilinear function under motion of the vertices. We work around this limitation by propagating identifiers to saddles in the rendering stage.

3.1.3 Tracking Algorithm:

We now present the various steps of our algorithm in more detail. The tracking driver controls the overall tracking behavior and it repeats the smoothing-tracking process to a time step specified by the user. A user typically would want to specify a time step large enough so that most of the noisy parts of the image have merged with dominant features. Here, *Image1* is the current state of the image and *Image2* is the image after smoothing.

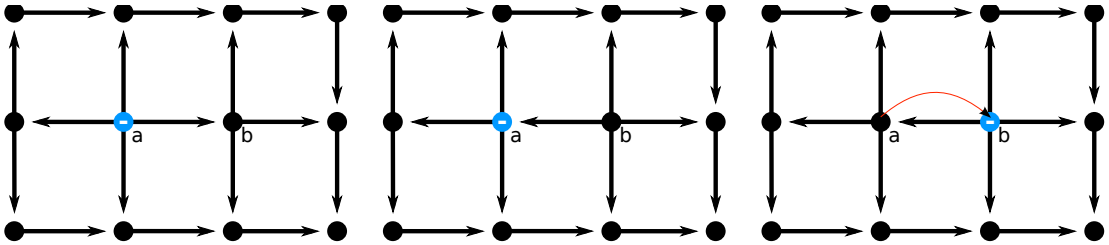
1) As one assigns *Image2*'s (smoothed image) values to *Image1* data element by data element:
   a) Get neighbors of current data element.
   b) Update current element's value in *Image1* to value in *Image2*.
   c) For each updated element, update all gradient arrows such that they point from the smaller to the larger neighbor.

The above steps ensure that at every point in the process the discrete vector field is a valid gradient field. Updating gradient arrows is done in the following manner:

1) We have two data elements: a and b and a gets updated. Flip the gradient arrow between them so that the gradient arrow points from b to a. We always do flips from the perspective of the element that used to be on the tail side of the arrow (a).
2) Test for the following conditions (not mutually exclusive):
   a) Data element a was a minimum before flipping gradient arrows but not one after flipping gradient arrows. Either it moved to b or canceled. Otherwise, determine whether a new minimum was created at b (anti-cancelation event).
   b) Data element a was not a maximum before flipping gradient arrows but is one after flipping gradient arrows. Determine if the maximum was previously at b and moved to a or was created (anti-cancelation event). Otherwise, determine if b was a maximum but was canceled.
3) Update the extremum status of a and b. They could have become regular points (i.e., non-extrema).

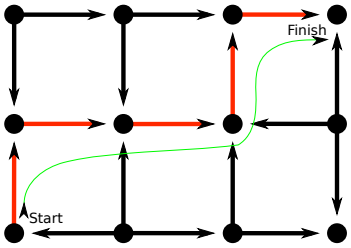
A visual depiction of the gradient arrow flipping process can be seen in Figure 5.

As mentioned previously, cancelation and anti-cancelation events require the walking uphill and downhill routines to pair with an extremum that already exists. We do this to merge points with the initial set of extrema that existed before

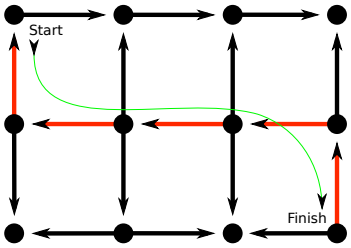


(a) Before flipping gradient arrow. (b) After flipping gradient arrow. (c) Label *a* moves to the correct vertex.

Fig. 5. When changing the value of point *a*, we have to recalculate the discrete gradient vectors. In this case, *a* is no longer a minimum as the gradient vector originally pointing from *a* to *b* flips. Since *b* is the location of a new minimum, the minimum has moved from point *a* to point *b*.

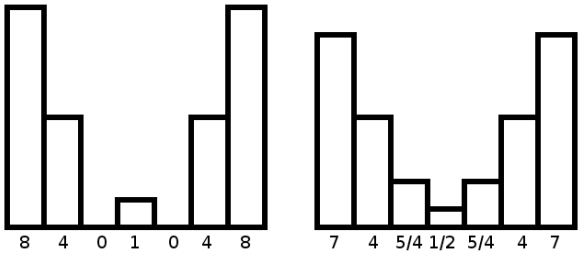


(a) Uphill walk example.



(b) Downhill walk example.

Fig. 6. Visual depiction of the walking uphill and downhill routines. In Figure 6(a), a maximum that was canceled at vertex "Start" has to pair with a parent maximum by walking in the direction of increasing values. As such, it walks in the direction of the gradient arrows to vertex "Finish." In Figure6(b), a minimum that was canceled at vertex "Start" needs to pair with a parent minimum so it walks against the direction of the gradient arrows to vertex "Finish." Pairing is done during the creation (anti-cancelation) and cancelation of extrema. Curved lines are provided to indicate the direction of walking.



(a) Before smoothing. (b) After smoothing.

Fig. 7. An image that is smoothed by the filter $\frac{1}{4}[1\,2\,1]$ (border values are duplicated for the filter). In Figure 7(a), we have two maxima on the border with one maximum in the center. The central maximum is smoothed out as shown in Figure 7(b). If one were to track the movement of extremum from the left to the right of the image while assigning the smoothed image's values to the original unsmoothed image, the smoothed maximum would follow the direction of the maximal gradient to pair with the leftmost maximum. This follows because when we reach the center, the values to the left are updated first and are steeper than the values to the right of the center. If one were to track from the right to the left, the smoothed maximum would pair with the rightmost maximum instead since the steeper values to the right of the center are updated before the values to the left. This artifact is due to the fact that one could arbitrarily pair the central maximum with either maximum that exist in the peripheries of the image. This problem is minor since it affects only one vertex.

smoothing began. An example of this process is illustrated in Figure 6. If we interpret smoothing as an additional temporal dimension, we will be able to follow the extrema that exist in the original data set over time, and we will be able to see them merge.

This method of tracking the movement of extrema requires us to modify the gradient arrows independently of each other. This is not possible when these arrows are defined implicitly by symbolic perturbation: changing the value of a voxel (which is necessarily discrete even when using floating-point representation) may flip multiple adjacent gradient arrows. In the scenario where an extremum moves to a neighbor, it would be difficult to track its movement when multiple (implicit) gradient arrows are flipped at the same time. Thus, we instead store the orientation of these arrows explicitly, which requires us to store at most three bits per vertex: one for the edge to the right, one for the edge behind, and one for the edge above a vertex. The bit indicates if the gradient arrow is orientated away from or toward that vertex.

We replace image values from the first to last pixel. In the ideal case one would not expect the order of image value replacement to affect tracking. Nevertheless, ambiguous cases can exist, as shown in Figure 7.

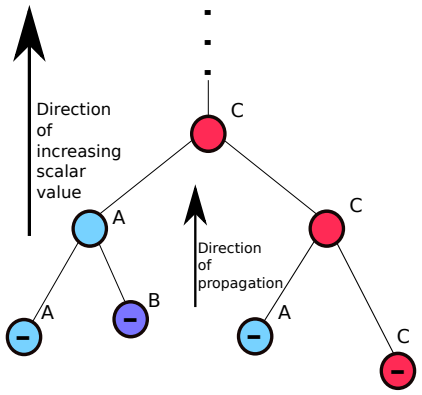


Fig. 8. An example of how we propagate labels from minima (which are marked with a minus symbol) to the saddles (not marked with a symbol) above them using the auxiliary join tree. Labels are indicated by colors and letters and these labels represent the features that exist at the current time step. The isovalue increases according to the arrow in the figure. As we move from the left side, the cyan label (A) is assigned to the first saddle because that label belongs to a "deeper minimum." From the right side, the red minimum (C) is propagated to the bottom-right saddle. The last saddle chooses the red label (C). Note that we propagate labels from the bottom up: the neighbors connected to a saddle must be labeled before it can be labeled.

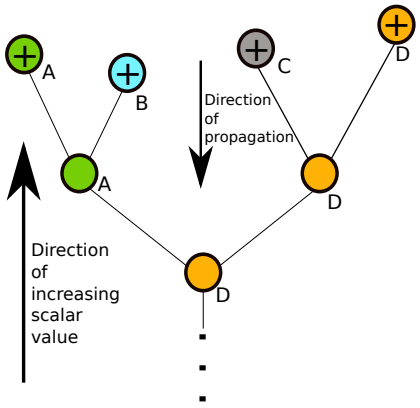


Fig. 9. An illustration showing how we propagate labels from maxima (which are marked with a plus symbol) to the saddles (not marked with a symbol) below them using the auxiliary split tree. This illustration is the same as the join tree version (Figure 8). However, it has an opposite orientation vertically and the labels are chosen based on higher maxima (which are more dominant). In this case, the maximum that originally has the orange label (D) influences more saddles than other maxima.

3.2 Rendering

After the tracking stage, we analyze our list of cancelation events in order to label portions of the unsmoothed data set's contour tree. Cancelations are useful to visualize how the regions implied by the topology of a data set merge after all the copies of a single label disappear. Thus, assuming that time again describes the number of smoothing and tracking steps, we can observe the changes in topology by using the cancelation events without destroying the data.

The volume renderer that we have developed has a time feature that merges components as implied by the topology. First, we assign a label to the original extrema of the contour tree, the topological data structure that is used by the volume renderer to define the transfer function. Each label corresponds to a hue, saturation, lightness (HLS) color, where the hue intensity is randomly picked, the saturation is full intensity and lightness is half intensity. This ensures that most (opaque) colors are fairly random, bright, and not washed-out. Afterwards, we use the cancelation events, up to a time step specified by the user, to merge extrema so that extrema that were smoothed out during the pre-processing stage share the same label as other extrema. Finally, we propagate the reduced number of labels that exist at this time step to the saddles of the contour tree, which may exist between extrema. To perform this step, we refer to the auxiliary join and split tree data structures which contain nodes that were used to construct the contour tree.

Since the saddles are not tracked during the tracking stage, we assign labels to them in a manner that reflects the principles underlying tracking. In other words, we merge relatively weaker features with stronger minima or maxima. The join tree, according to a definition similar to the one found in Carr *et al.* [4], represents the merging of components from the minima up to saddles. During the rendering stage this auxiliary tree has a limited number of labels (due to the extrema-merging process), and we propagate the labels of dominant, or "deeper," minima to saddles. An example of this process is shown for the join tree in Figure 8.

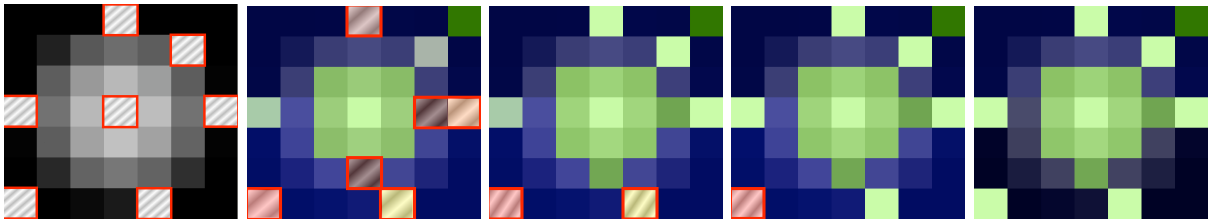
The split tree, on the other hand, represents the splitting of components from saddles to the maxima. In this case we give preference to larger maxima when propagating labels to the saddles below them. In other words, preference is given to maxima that belong to higher "peaks" because they are more resistant to smoothing than noisy maxima. The example for this process on a split tree is shown in Figure 9.

After this labeling operation, we assign color values to voxels based on where a voxel exists in an arc of a contour tree. If it exists in an arc with a single label, it will receive that label's color. If it exists inside an arc with two labels, it will receive an interpolated color that is derived from those two labels' colors.

3.3 2D Examples

Before discussing our results, we illustrate how our tracking and rendering procedures can be used to affect two examples: two two-dimensional height maps represented as gray-scale images. Thus, the brighter a pixel is, the higher it is in value and white pixels represent maxima or "peaks." In the following "mountain-like" examples, there exist spurious peaks that make the images noisy. We demonstrate how these "maximal regions" merge under the influence of a smoothing operator.

To make these illustrations easier to understand, we assigned a randomly generated color, or label, to each maximum. Re-



(a) Normal image (b) Normal image, labeled (c) Time step 1, labeled (d) Time step 2, labeled (e) Time step 5, labeled

Fig. 10. A two-dimensional height map represented as a gray-scale image. Brighter areas indicate higher values. The data consists of a dominant peak (largest maximum) in the center that is surrounded by six other peaks. All peaks are highlighted in Figure 10(a) and have a gradient texture. Initially, each peak and the regions around the peaks have a unique color (i.e., label), whereas valleys are indicated in blue. Pixels between two regions are assigned a color that is interpolated between those two regions' colors (with bias toward the "higher" region). As the image is progressively smoothed and tracked as one moves forward in time, the coloring indicates how different regions merge, as evidenced by the generation of a single color. This example illustrates how weaker features (highlighted from Figure 10(b) to Figure 10(e) and identified by a gradient texture) merge with stronger ones.

gions that have merged with a maximum share the maximum's color. Since our focus for these 2D examples is to study features merging in relation to maxima, the "minimal regions" are assigned a dark blue color which is less distracting than the regions corresponding to maxima. The time step associated with these pictures indicates how many times an image was smoothed before being labeled. If a pixel is between two extremal regions, we generate a color which is the result of interpolation of the two extrema's colors (with bias toward the "higher" region). See Figures 10 and 11.

The tracking and smoothing procedures are transparent processes to a user of our program. They produce the colored results that can be explored and analyzed by the user.

In Figure 10, we show the image of a single maximum (shown as a central peak) which is surrounded by other maxima. In Figure 10(b) and Figure 10(a), we have included pictures of the unsmoothed image with and without labeling, respectively. Note the number of spurious maxima (highlighted in red and textured) in the labeled screenshot. In the following images, however, one can see that due to repeated smoothing, the number of labeled regions (indicated by color) is being reduced. Eventually, all maxima merge and share the same (green) label.

A similar result is obtained for the example shown in Figure 11. Initially, the two peak maxima are surrounded by four maxima which cause the labeling to appear to be less uniform. As one smoothes substantially, the spurious regions merge around the two peaks. Eventually, the image is over-labeled as both peaks share the same label. Thus, Figure 11(d) represents high-quality labeling.

Both of these examples illustrate an important point: our tracking and rendering algorithms are designed to merge weaker features with stronger ones by applying a labeling operation. Once the weaker features are assigned the same label as their stronger associated parent label, we are able to label the underlying (unsmoothed, noisy) contour tree's nodes, which influences the resulting visualization.

4 Results and Discussion

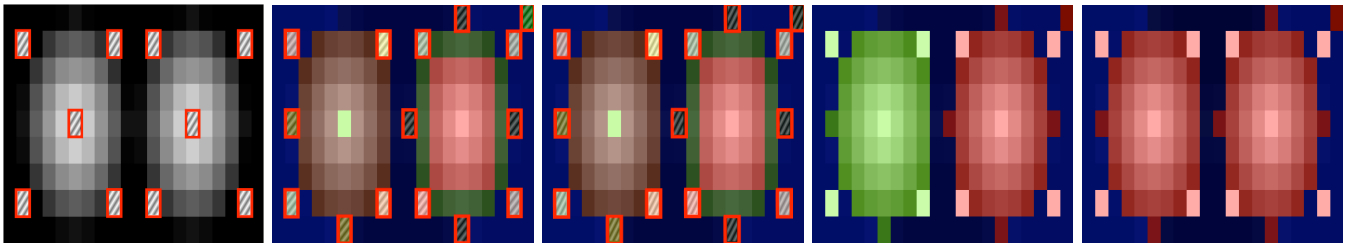
We discuss how our tracking and rendering algorithms behave when applied to 3D, volume data and discuss how it may apply to transfer function design. We use a tracker and volume renderer written in C++. The latter application was developed using the QT and OpenGL libraries. It employs a GPU ray caster based on the techniques covered in [13]. In the volume render interface, the user is able to use a spinbox to modify the merging parameter to label (color) the image into different regions. Whenever the user selects a merging level, the application computes and displays regions that exist up to that merging level. We also allow the user to modify a global opacity transfer function and render isosurfaces to provide context for the merging results. The opaque colors generated are based on the labels produced by our topological analysis algorithm. An example of the program's interface is shown in Figure 12.

The tracker and volume rendering applications were performed on an Intel Core i7 950, with 6 GB of RAM and an NVIDIA GeForce GTX 260 896MB video card. We applied our methods to the head MRI CISS 8Bit (256x256x124), silicium (98x34x34), and the x-ray scan of a human foot (256x256x256) data sets. The head MRI data set highlights the cerebro-spinal-fluid cavities of the head. The silicium data set is the result of a simulation of a silicium grid and was included to see how our methods label a simulated data set. The foot data set is a rotational c-arm x-ray scan of a human foot and contains tissue and bone data. All data sets were obtained from `http://volvis.org`.

4.1 Volume Data Examples

We now discuss how our algorithm is able to label interesting structures in various data sets. We describe when our algorithm begins to over-label the images.

Foot data set: In the first set of rendering results, we experimented with the foot data set after assigning low opacity to the skin and tissue portions of the foot by modifying the global opacity transfer function (see Figure 12). In Figure 13(a), the bones are not clearly identified and appear to be quite



(a) Normal image (b) Normal image, labeled (c) Time step 1, labeled (d) Time step 2, labeled (e) Time step 22, labeled

Fig. 11. A two-dimensional height map represented by gray-scale color. All peaks are highlighted in Figure 11(a) and are textured. Like the image with one peak in the center, regions around peaks are assigned a color (i.e., label) while low-value regions are assigned a uniform blue color. Unlike the example shown in Figure 10, the example shown here contains two dominant peaks. The different extremal regions quickly merge after the program smoothes the image and tracks the movement of extrema in the background. Noisy regions are highlighted from Figure 11(b) to Figure 11(e) and have a gradient texture.

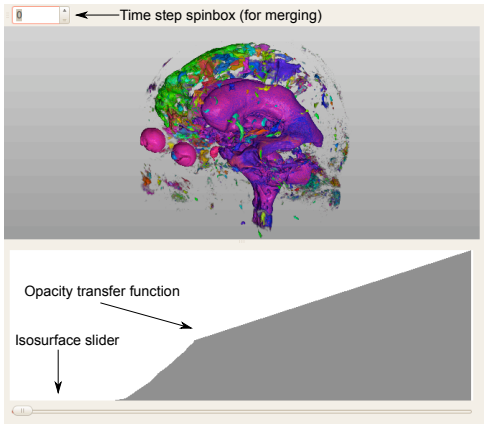


Fig. 12. Volume renderered head MRI dataset with interface. The spinbox near the top of the window allows the user to control the level of merging by specifying the last time step to do merging to. The global opacity transfer function can be modified by dragging the left mouse bottom over the graph region, where the x-axis represents scalar value (from least to most dense) while the y-axis is intensity (from transparent to opaque). The (opaque) colors created are based on a transfer function generated by our topological analysis algorithm. The isosurface slider, which can be moved across increasing scalar values, lies at the bottom of the window and can be used to draw a slightly gray isosurface based on the isovalue specified.

noisy. Figure 13(b) to Figure 13(d) illustrate the changes in the regions identified in the bones. In Figure 13(b), one can see regions split into two portions; one portion belongs to the volume containing the large, metatarsal bones and the other portion contains the smaller phalanges. Over time, these portions merge, as seen in Figure 13(c). In Figure 13(d), the image starts to become over-labeled as the separate bones of the foot begin to the share the same color.

In this case, it is difficult to say which time step is "good." The user can change the time step and decide which visualization provides the best representation of the data set.

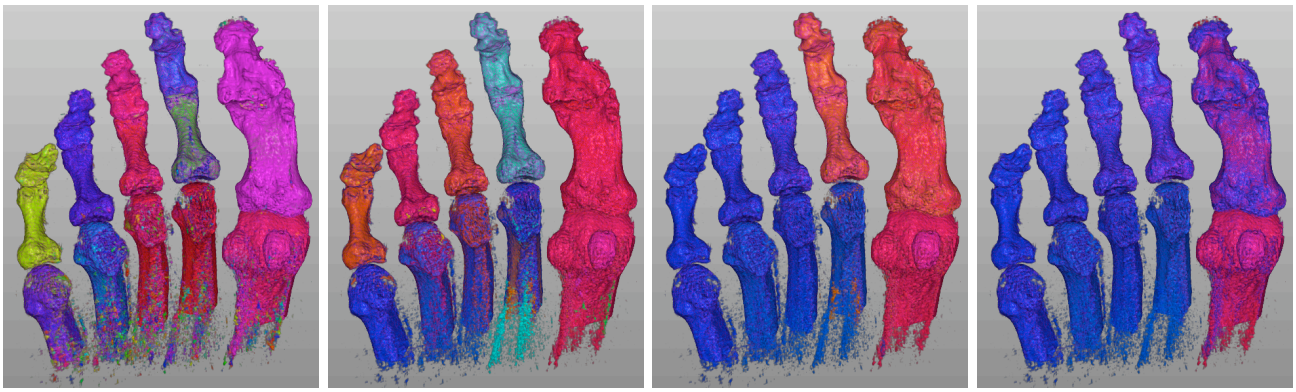
Head MRI data set: In Figure 14, we include results for the MRI head data set. Similar to the results for the foot data set, these screenshots illustrate how smaller regions tend to cluster around a single label as one moves forward in time. In this case, we modified the global opacity transfer function to help illustrate how the cerebral ventricles of the brain are rendered. In Figure 14(a) and Figure 14(b), the ventricle portion of the brain appears very noisy. Nevertheless, it becomes more uniform in color after one increases the time step count to 320, see Figure 14(c). One must be careful not to increase the time step too much; in Figure 14(d) the time step is increased to 450, which causes a portion of the nearest eye to share the same color as nearby regions.

Silicium data set: Even though this data set does not contain a significant amount of noise, it is worth studying because quantization noise and the discretization create spurious topological features that influence standard contour-tree-based approaches. We examine it to demonstrate how our method merges topological features that are clearly and obviously separate before smoothing. In the set of pictures illustrating simulated electron density in a silicium grid, shown in Figure 15, one can see the behavior of electron bonds in relation to an increasing time step. As the time step increases, bonds which lie close to each other quickly merge and share the same label. In Figure 15(d), merging is continued to the point where it becomes less informative. By allowing the user to label the data set beyond the point of the labeling being reasonable we make it possible to pick an appropriate degree of merging in an interactive, exploratory process.

4.2 Tracking Statistics

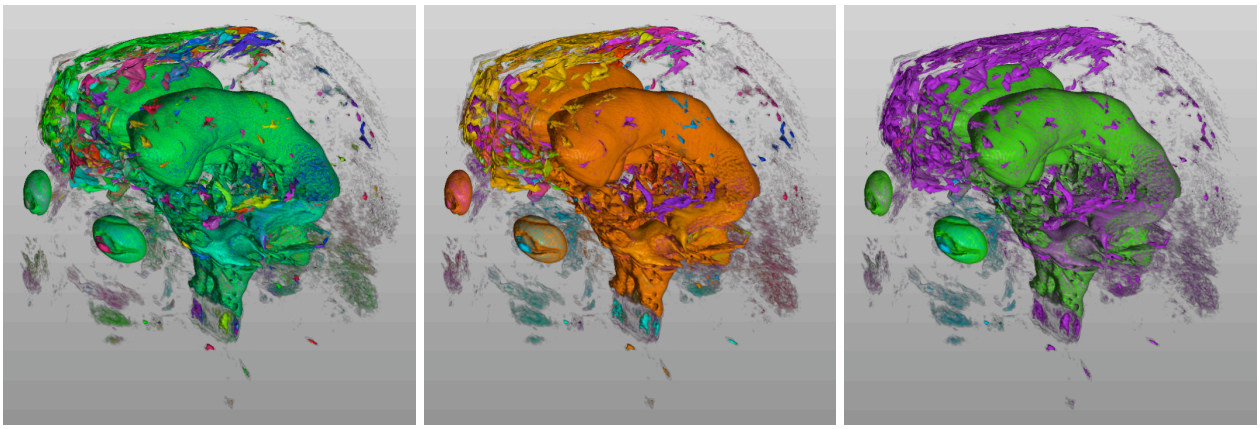
In Table 1, we provide times required by the tracker to process each data set after an indicated number of time steps. While some of the times in this table are fairly large, especially for the foot and head data sets, one must keep in mind that the tracking process needs to be done once before the tracking results are used for rendering. In the rendering stage, the user can observe the visual changes of merging from one time step to the next.

In Table 2, we provide the number of labels (of extrema) once the tracker has iterated for a certain number of time steps.



(a) Foot labeling after 0 time steps. (b) Foot labeling after 128 time steps. (c) Foot labeling after 380 time steps. (d) Foot labeling after 400 time steps.

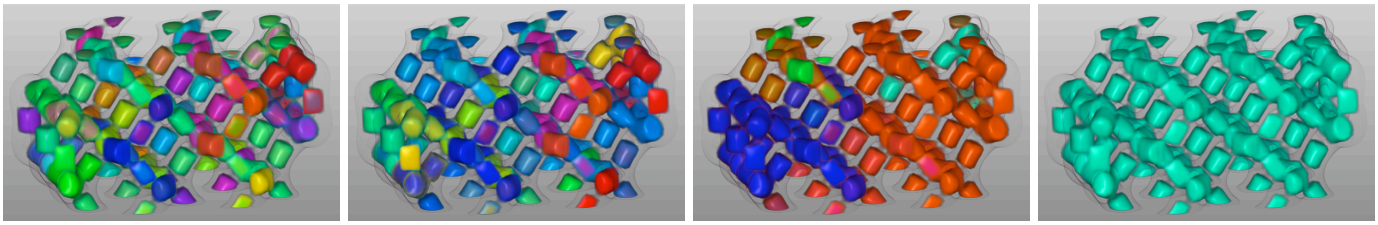
Fig. 13. Labeling of different regions of a computer tomography scan of a foot. The global opacity transfer function has been chosen to reveal the bones (see Figure 12). While the original image (13(a)) is labeled into noisy clusters, with continuous merging the bone structures are found leading to identification of individual bones around time step 128. When we continue merging, neighboring bones are merged leading to the point where individual toes are grouped around time step 380. At time step 400, the merging process starts grouping toes into single clusters.



(a) Head labeling after 0 time steps. (b) Head labeling after 64 time steps. (c) Head labeling after 320 time steps.

(d) Head labeling after 450 time steps.

Fig. 14. Head MRI data set. The global opacity transfer function has been modified to highlight the changes in the cerebral ventricles as one moves forward in time. It can be seen that our algorithm removes most of the initial noise leading to a labeling of the eyes, the ventricle, and the brain stem in time step 320. After this iteration, the eyes begin to merge with nearby regions leading to over-labeling.



(a) Silicium labeling after 0 time steps. (b) Silicium labeling after 16 time steps. (c) Silicium labeling after 32 time steps. (d) Silicium labeling after 256 time steps.

Fig. 15. Simulated electron density in a silicium grid. The global opacity transfer function is chosen to highlight the grid structure, especially the electron bonds. Our approach initially assigns a different label to each electron bond in the data set, but with increased levels of merging, some bonds are grouped because they lie closer together than other bonds which shows minor irregularities in the grid. At time step 256, most of the dataset shares the same label and no more merging is possible.

TABLE 1
Timing results for smoothing and tracking up to a certain time step, which are both pre-processing steps computed on the Intel i7 950 processor with a single thread. The first row contains columns indicating the time step up to which smoothing and tracking was performed. Columns to the right of each data set indicate how long pre-processing took, which needs to be done only once before analyzing a data set.

Data Set	16	32	64	128	256	512
Foot	1 min, 54 sec	3 min, 16 sec	5 min, 46 sec	10 min, 34 sec	22 min, 41 sec	44 min, 14 sec
Head MRI	46 sec	1 min, 31 sec	2 min, 43 sec	5 min, 1 sec	10 min, 35 sec	20 min, 25 sec
Silicium	0.8 sec	1 sec	3 sec	5 sec	9 sec	18 sec

TABLE 2
Data sets and the number of distinct labels at a given time step. It can be seen that the numbers of labels decreases with increased smoothing and most of the labels already disappear after the first sixteen smoothing steps. The corresponding visualizations are shown in Figures 13, 14, and 15.

Time Steps	Labels Foot	Labels head MRI	Labels Silicium
0	606226	1733964	283
16	1803	1492	46
32	600	495	12
64	196	171	3
128	63	56	3
256	25	21	2
512	7	9	2

As expected, the number of labels decreases when the tracker applies more smoothing and merging steps. One interesting characteristic about the silicium data set is the fact that the rate at which the number of labels decreases is relatively high. Since this data set is fairly simple and contains smaller structures, this can be explained by the fact that most of the data becomes "flat" during early stages of smoothing.

In addition to timing and label statistics, we provide the number of labeled nodes in Table 3. The total number of nodes in the contour tree for each data set is the sum of the number of minima, maxima, and saddles. With increased smoothing, the number of labeled nodes remains the same; however, their labels become less diverse due to merging.

TABLE 3
Data sets with critical point counts. Our algorithm labeled the same number of critical points for each time step. The difference in each time step lies in the number of labels used.

Data set	Minima	Maxima	Saddles
Foot	279298	326928	606221
Head MRI	891812	842152	1733918
Silicium	143	140	281

4.3 Discussion

As our results illustrate, our algorithm is able to isolate interesting structures based on topological information in each data set. The user can guide the labeling by choosing which level of merging of labels is appropriate. While the results are not perfect, one must keep in mind that the results illustrated here require little input from the user and no additional domain knowledge and can be generated with a robust and straightforward approach.

The labeling process and transfer function creation process is fairly responsive for most data sets and takes about one to two seconds. On some large volumes, such as the foot, the user must wait about four seconds to see an update from the transfer function after choosing a time step. For the MRI data set, this time can increase to up to twelve seconds.

4.4 Transfer Function Design

Since our topological analysis method automatically creates a transfer function that assigns a color to voxels based on their locations in the clustered contour tree, we believe that a

viable application of our technique is transfer function design. We have compared our results with those obtained with the transfer functions described in Weber and Scheuermann [41], which had originally been developed by Fujishiro *et al.* [42].

The results of these transfer functions are shown in Figure 16. Figures 16(a) to 16(c) were generated using a transfer function that emphasizes topologically equivalent isosurfaces. Figures 16(d) to 16(f) use a transfer function emphasizing regions close to critical isovalues. Our results are shown in in a third row, in Figures 16(g) to 16(i).

The two types of transfer functions, topological equivalence and proximity to critical isovalues, generate visualizations that are quite similar. As expected, the latter tends to highlight voxels that are close to or correspond to critical isovalues. While our method makes all regions of the data opaque and requires one to use a global opacity transfer function, these two transfer functions automatically assign low opacity values to regions of less relevance. Unlike our approach, however, these custom transfer functions do not take local features into account. For instance, our method colors different regions in the foot based on their proximity to specific bones, as can be seen in Figure 16(g). In Figure 16(a) and Figure 16(d), all of the foot's bones are colored in a similar manner. A similar statement holds true for the MRI head and silicium data sets where it is also difficult to isolate regions based on this type of transfer function.

We believe that our method is useful for more general transfer function design as it permits the definition of a transfer function that is not globally defined. Instead, it applies color values based on voxel location inside a contour tree, which contains labels defining colors of different nodes. Once nodes are clustered together and share the same color, the effective transfer function becomes less noisy.

5 Conclusions and Directions for Future Research

We have introduced a topological analysis method that is designed to handle noise but does not require one to use a time-varying topological data structure. We track topological events during a pre-processing stage as an image is convolved with a smoothing filter multiple times, and we use these events to label the image's contour tree. This labeling procedure merges portions of the data set in an attempt to reduce the effects of noise in the resulting visualization. Although most steps of this process are automatic and require a user to specify only a few parameters, we allow the user to control the level of labeling by merging features up to a certain time step.

Our approach combines data smoothing, topological data analysis and characterization, and interactive data exploration. While topological data analysis is an extremely powerful approach to understanding qualitative, structural properties and their relationships, it is in general impossible to devise a fully automatic method for simplifying the topology of a given scalar-valued data set to a level of simplicity that is meaningful and appropriate for a specific application. User control and involvement in a smoothing-based topological data simplification is crucial to draw proper scientific conclusions from a complex scientific data set. We devised our method having these objectives in mind.

One can consider various directions for improving our approach. The transfer function generator procedure in the volume renderer is the current bottleneck and can make the labeling procedure slow. This function could be optimized as it is independent from our current merging algorithm. Additionally, one could consider finding ways to make our tracking results independent from the order of tracking.

Besides modifying significant portions of the analysis algorithm, one could allow a user to influence the merging process directly. For instance, our system currently does not allow the user to identify portions of the data set that are necessary or noisy (unnecessary) portions. This feature can be implemented via an interactive contour tree interface. It is desirable to investigate this approach by trying to find a balance between a fully automatic and user-controllable algorithm.

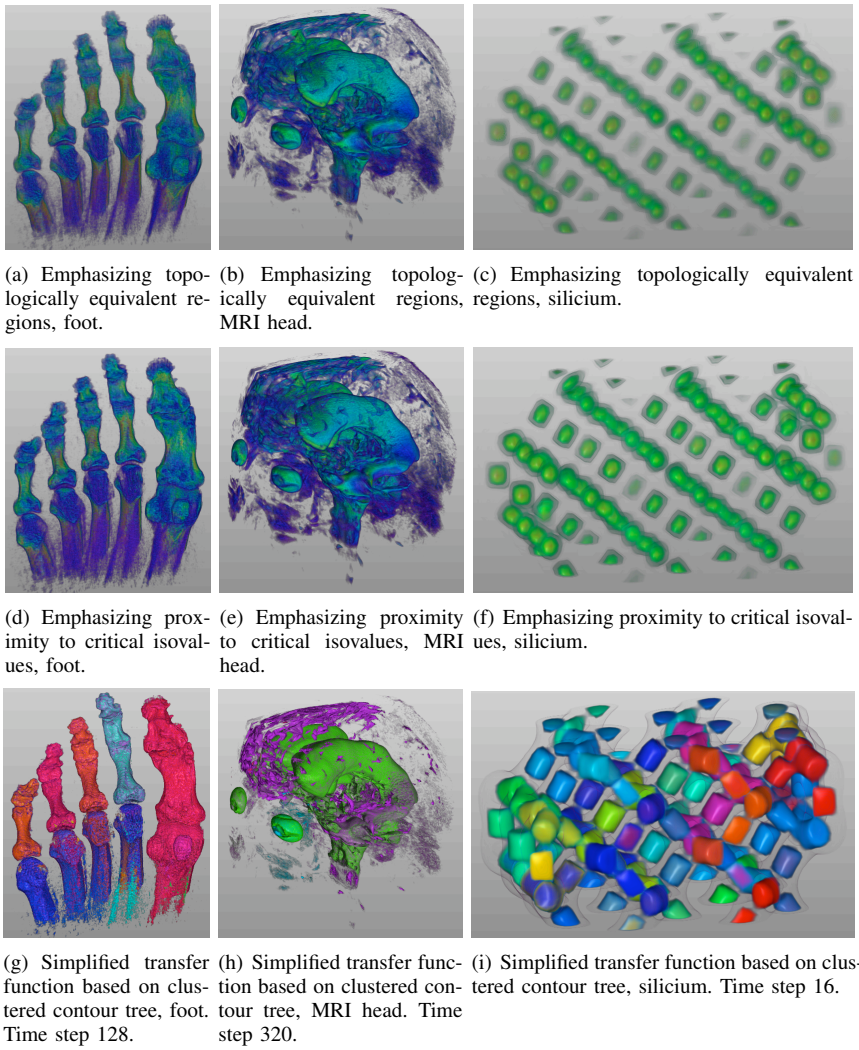
Another possibility for future research is the refinement of our approach for noisy materials science data sets (image data sets), which would require one to reconstruct nanoporous metals as shown in [43]. These data sets can exhibit substantial levels of noise, and we believe that this direction of research might lead to additional relevant improvements and applications of our methods.

Acknowledgements

This work was supported by the Materials Design Institute, funded by the UC Davis/Los Alamos National Laboratory Educational Research Collaboration (LANL Agreement No. 25110-002-06), and the UC Lab Fees Research Program Contingency Funds. In addition, this effort was supported by the National Science Foundation through grant CCF-0702817. We thank our colleagues from the Institute for Data Analysis and Visualization (IDAV), Department of Computer Science, UC Davis and Gunther Weber for providing code from his publication [41].

References

[1] J. Milnor, *Morse theory*. Princeton university press, 1963.
[2] G. Reeb, "Sur les points singuliers d'une forme de pfaff complètement intégrable ou d'une fonction numérique," *Comptes Rendus de l'Acadèmie des Sciences de Paris*, vol. 222, pp. 847–849, 1946.
[3] R. L. Boyell and H. Ruston, "Hybrid techniques for real-time radar simulation," in *Proceedings of the 1963 Fall Joint Computer Conference*. IEEE, 1963, pp. 445–458.
[4] H. Carr, J. Snoeyink, and U. Axen, "Computing contour trees in all dimensions," *Computational Geometry – Theory and Applications*, vol. 24, no. 2, pp. 75–94, Feb. 2003.
[5] Y. Chiang, T. Lenz, X. Lu, and G. Rote, "Simple and optimal output-sensitive construction of contour trees using monotone paths," *Computational Geometry: Theory and Applications*, vol. 30, no. 2, pp. 165–195, 2005.
[6] V. Pascucci and K. Cole-McLaughlin, "Parallel computation of the topology of level sets," *Algorithmica*, vol. 38, no. 2, pp. 249–268, Oct. 2003.
[7] V. Pascucci, G. Scorzelli, P. Bremer, and A. Mascarenhas, "Robust on-line computation of Reeb graphs: simplicity and speed," in *SIGGRAPH International Conference on Computer Graphics and Interactive Techniques*. ACM New York, NY, USA, 2007.
[8] Y. Shinagawa, T. L. Kunii, and Y. L. Kergosien, "Surface coding based on morse theory," *IEEE Comput. Graph. Appl.*, vol. 11, no. 5, pp. 66–78, 1991.



(a) Emphasizing topologically equivalent regions, foot.
(b) Emphasizing topologically equivalent regions, MRI head.
(c) Emphasizing topologically equivalent regions, silicium.
(d) Emphasizing proximity to critical isovalues, foot.
(e) Emphasizing proximity to critical isovalues, MRI head.
(f) Emphasizing proximity to critical isovalues, silicium.
(g) Simplified transfer function based on clustered contour tree, foot. Time step 128.
(h) Simplified transfer function based on clustered contour tree, MRI head. Time step 320.
(i) Simplified transfer function based on clustered contour tree, silicium. Time step 16.

Fig. 16. Results based on transfer function design techniques described by Weber and Scheuermann [41] and our method. We applied an additional global opacity transfer function to these methods to match our program's results. The transfer functions used in Figures 16(a) through 16(c) emphasize topologically equivalent regions. The transfer functions used in Figures 16(d) through 16(f) emphasize proximity to critical isovalues. Our method is shown in Figures 16(g) through 16(i).

[9] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. R. Schikore, "Contour trees and small seed sets for isosurface traversal," in *Proceedings of the 13th ACM Annual Symposium on Computational Geometry (SoCG)*. ACM Press, 1997, pp. 212–220.

[10] C. L. Bajaj, V. Pascucci, and D. R. Schikore, "The contour spectrum," in *Proc. IEEE Visualization '97*, R. Yagel and H. Hagen, Eds. New York, New York: ACM Press, Oct. 19–24 1997, pp. 167–173.

[11] H. Carr and J. Snoeyink, "Path seeds and flexible isosurfaces using topology for exploratory visualization," in *Data Visualization 2003 (Proceedings VisSym 2003)*. New York, NY: ACM Press, 2003, pp. 49–58.

[12] S. Takahashi, Y. Takeshima, and I. Fujishiro, "Topological volume skeletonization and its application to transfer function design," *Graphical Models*, vol. 66, no. 1, pp. 24 – 49, Jan. 2004.

[13] G. H. Weber, S. E. Dillard, H. Carr, V. Pascucci, and B. Hamann, "Topology-controlled volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 2, pp. 330–341, 2007.

[14] P.-T. Bremer, H. Edelsbrunner, B. Hamann, and V. Pascucci, "A topological hierarchy for functions on triangulated surfaces," *IEEE Transactions Visualization and Computer Graphics*, vol. 10, no. 4, pp. 385–396, 2004.

[15] A. Gyulassy, V. Natarajan, V. Pascucci, and B. Hamann, "Efficient computation of morse-smale complexes for three-dimensional scalar functions," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1440–1447, 2007.

[16] A. Gyulassy, M. Duchaineau, V. Natarajan, V. Pascucci, E. Bringa, A. Higginbotham, and B. Hamann, "Topologically clean distance fields," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1432–1439, 2007.

[17] D. Laney, P. Bremer, A. Macarenhas, P. Miller, and V. Pascucci, "Understanding the structure of the turbulent mixing layer in hydrodynamic instabilities," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, p. 1053, 2006.

[18] L. Lifshitz and S. Pizer, "A multiresolution hierarchical approach to image segmentation based on intensity extrema," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 6, pp. 529–540, 1990.

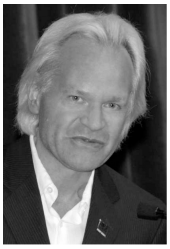
[19] L. Florack and A. Kuijper, "The topological structure of scale-space images," *Journal of Mathematical Imaging and Vision*, vol. 12, no. 1, pp. 65–79, 2000.

[20] A. Kuijper and L. Florack, "The hierarchical structure of images," *IEEE Transactions on Image Processing*, vol. 12, no. 9, pp. 1067–1079, 2003.

[21] Y. Gingold and D. Zorin, "Controlled-topology filtering," in *Proceedings of the 2006 ACM symposium on Solid and physical modeling*. ACM, 2006, p. 61.

[22] A. Szymczak, "Subdomain aware contour trees and contour evolution in time-dependent scalar fields," *International Conference on Shape*

Modeling and Applications, pp. 136–144, 2005.
[23] H. Edelsbrunner, J. Harer, A. Mascarenhas, V. Pascucci, and J. Snoeyink, "Time-varying Reeb graphs for continuous space–time data," *Computational Geometry: Theory and Applications*, vol. 41, no. 3, pp. 149–166, 2008.
[24] H. Edelsbrunner and J. Harer, "Jacobi sets of multiple Morse functions," *Foundations of Computational Mathematics, Minneapolis 2002*, p. 37, 2004.
[25] B. Sohn and C. Bajaj, "Time-varying contour topology," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 1, pp. 14–25, 2006.
[26] D. Silver and X. Wang, "Tracking scalar features in unstructured datasets," in *VIS '98: Proceedings of the conference on Visualization '98*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1998, pp. 79–86.
[27] J. Chen, D. Silver, and L. Jiang, "The feature tree: Visualizing feature tracking in distributed amr datasets," in *PVG '03: Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*. Washington, DC, USA: IEEE Computer Society, 2003, p. 14.
[28] I. Fujishiro, R. Otsuka, S. Takahashi, and Y. Takeshima, "T-map: A topological approach to visual exploration of time-varying volume data," in *High-Performance Computing*. Springer, 2009, pp. 176–190.
[29] P. Keller and M. Bertram, "Modeling and visualization of time-varying topology transitions guided by Hyper Reeb Graph structures," in *Proceedings of the Ninth IASTED International Conference on Computer Graphics and Imaging*. Citeseer, 2007, pp. 15–20.
[30] S. Takahashi, Y. Kokojima, and R. Ohbuchi, "Explicit control of topological transitions in morphing shapes of 3D meshes," in *Proceedings of the 9th Pacific Conference on Computer Graphics and Applications*. Citeseer, 2001, pp. 70–79.
[31] T. Nieda, A. Pasko, and T. L. Kunii, "Detection and classification of topological evolution for linear metamorphosis," *The Visual Computer: International Journal of Computer Graphics*, vol. 22, no. 5, pp. 346–356, 2006.
[32] T. Lindeberg, *Scale-Space Theory in Computer Vision*, 1994.
[33] H. Carr, J. Snoeyink, and M. van de Panne, "Simplifying flexible isosurfaces using local geometric measures," in *VIS '04: Proceedings of the conference on Visualization '04*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 497–504.
[34] S. Takahashi, I. Fujishiro, and Y. Takeshima, "Interval Volume Decomposer: A Topological Approach to Volume Traversal," in *Proceedings of Visualization and Data Analysis 2005*, 2005, pp. 103–114.
[35] Takahashi, S. and Nielson, G. M. and Takeshima, Y. and Fujishiro, I., "Topological volume skeletonization using adaptive tetrahedralization," in *GMP '04: Proceedings of the Geometric Modeling and Processing 2004*. Washington, DC, USA: IEEE Computer Society, 2004, p. 227.
[36] H. Carr, T. Möller, and J. Snoeyink, "Simplicial subdivisions and sampling artifacts," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 2, pp. 231–242, 2006.
[37] G. M. Nielson, "On marching cubes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 3, pp. 341–351, Jul.–Sep. 2003.
[38] H. Edelsbrunner, J. Harer, A. Mascarenhas, and V. Pascucci, "Time-varying reeb graphs for continuous space-time data," in *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*. New York, NY, USA: ACM, 2004, pp. 366–372.
[39] H. Carr and N. Max, "Subdivision Analysis of the Trilinear Interpolant," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, pp. 533–547, 2009.
[40] J. C. Russ, *The Image Processing Handbook, Fifth Edition*. Boca Raton, FL, USA: Taylor & Francis Group, LLC, 2007.
[41] G. H. Weber and G. Scheuermann, *Automating Transfer Function Design Based on Topology Analysis*, 2004.
[42] I. Fujishiro, Y. Takeshima, T. Azuma, and S. Takahashi, "Volume data mining using 3D field topology analysis," *IEEE Computer Graphics and Applications*, pp. 46–51, 2000.
[43] H. Rösner, S. Parida, D. Kramer, C. Volkert, and J. Weissmüller, "Reconstructing a nanoporous metal in three dimensions: An electron tomography study of dealloyed gold leaf," *Advanced Engineering Materials*, vol. 9, no. 7, pp. 535–541, 2007.



Bernd Hamann studied mathematics and computer science at the Technical University of Braunschweig, Germany, and Arizona State University. Since 1995 he has been affiliated with the University of California, Davis, where his research and teaching efforts have focused on visualization, geometric modeling and computer graphics.

Mario Hlawitschka is a postdoctoral researcher at the Institute for Data Analysis and Visualization and the Department of Biomedical Engineering at the University of California, Davis. He studied Computer Sciences and Electrical Engineering at the University of Kaiserslautern, Germany with a focus on signal processing and visualization of tensor data and received an MSc in Computer Science in 2004. He received his PhD in Computer Science from the Universitt Leipzig, Germany, in 2008 for his work on the visualization of magnetic resonance tomography data. His current research focuses on coherent structures in tensor fields, hardware-based acceleration of visualization and simulation algorithms, topological- and comparative visualization for various applications such as the geosciences, neuroscience and neuro-surgery, and bioengineering.

Scott Dillard received the PhD degree from the University of California, Davis in 2009. Since 2010 he has been a staff scientist at Pacific Northwest National Laboratory in the Visual Analytics group, and before that he was a computer science researcher in the Materials Design Institute at Los Alamos National Laboratory working in collaboration with the Institute for Data Analysis and Visualization (IDAV) at UC Davis. His research interests include computer graphics and visualization, computer vision, computational geometry and topology.

Sohail Shafii received his bachelor's degree from University of California, Davis in 2006. He is currently a Computer Science PhD student at University of California, Davis, working for the Institute for Data Analysis and Visualization (IDAV) and Los Alamos National Laboratory (LANL). His research background consists of visualization, computer graphics, point cloud processing, topology, and feature detection.