Published in final edited form as:
IEEE Trans Vis Comput Graph. 2006 ; 12(1): 14–25. doi:10.1109/TVCG.2006.16.

Time-Varying Contour Topology

Bong-Soo Sohn and
Department of Computer Sciences, TAY 2.124, The University of Texas at Austin, 1 University Station C0500, Austin, TX 78712-1188. E-mail: bongbong@cs.utexas.edu.

Chandrajit Bajaj
Center for Computational Visualization, Department of Computer Sciences, and the Institute for Computational Engineering and Sciences, ACES 2.324, The University of Texas at Austin, Austin, TX 78712-1188. E-mail: bajaj@cs.utexas.edu.

Abstract

The contour tree has been used to compute the topology of isosurfaces, generate a minimal seed set for accelerated isosurface extraction, and provide a user interface to segment individual contour components in a scalar field. In this paper, we extend the benefits of the contour tree to time-varying data visualization. We define temporal correspondence of contour components and describe an algorithm to compute the correspondence information in time-dependent contour trees. A graph representing the topology changes of time-varying isosurfaces is constructed in real-time for any selected isovalue using the precomputed correspondence information. Quantitative properties, such as surface area and volume of contour components, are computed and labeled on the graph. This topology change graph helps users to detect significant topological and geometric changes in time-varying isosurfaces. The graph is also used as an interactive user interface to segment, track. and visualize the evolution of any selected contour components over time.

Keywords

Contour tree; level set topology; feature tracking; time-varying volume visualization

1 Introduction

Scientific simulations of today often generate large time-varying scalar fields (i.e., real-valued functions). Computational visualization techniques use modeling and rendering methods to aid scientific discovery and calibration of simulations. This involves identification, extraction, and quantitative analysis of features present in data which are then visualized. Isosurface extraction or volume rendering is a common way to visualize the evolution of features in data. However, just rendering a sequence of volumes or isosurfaces (i.e., level sets) does not explicitly yield an effective visualization of its dynamic features. In this paper, we describe an algorithm to compute correspondence information of contours for all isovalues in time-varying scalar fields. This allows us to interactively track the topology changes of time-varying isosurfaces and to extract additional quantitative information. For instance, in a cosmological simulation which generates time-varying density and temperature fields in the universe providing information about the formation of galaxy clusters, we are able to detect when and where an individual galaxy cluster forms, disappears, splits, and merges with other clusters. In addition, we can measure the extent to which a galaxy cluster grows or shrinks. The images and videos of the data are available on our Website [11]. Other examples of feature tracking can be found in [16], [21].

The *Contour Tree* (CT) [6], [18] is useful for the visualization of a scalar field. First, CT provides topological information of a scalar field which is not directly obtained from rendering techniques. Second, CT generates a minimal seed set for efficient isosurface extraction [1], [27]. Third, CT provides a user interface to segment and render each individual connected component of an isosurface. However, CT is used for a single scalar field and its benefits are limited to the visualization of a static scalar field. Our main objective is to extend the benefits of CT to time-varying data visualization.

The input to our approach is a time-varying scalar field $\{f^1, \ldots, f^T\}$, where f^t is a piecewise linear (PL) function defined on the same simplicial domain mesh for each time t. We constrain the domain to be a simply connected three-dimensional volume. A contour tree CT^t is constructed for each time t and embedded on a 2D plane, such that the y coordinate of each node in CT^t is its function value. The x coordinate of a node can be any arbitrary value. When an isovalue w is selected, each *intersection point* p_k^t between CT^t and a line $y = w$ represents a connected component C_k^t of an isosurface. We call a connected component of an isosurface a *contour*.

Now, we consider two contour trees, CT^t and CT^{t+1}, and an arbitrary isovalue w. Intersection points can be obtained from the contour trees. We represent correspondence information with $p_k^{t+1} \leftarrow \{p_1^t, \ldots, p_n^t\}$, meaning that each of $C_1^t, \ldots, C_n^t$ corresponds (evolves) to C_k^{t+1}. Colors in Figs. 1a and 1b depict the correspondence relationship. Since w is an arbitrary value in an interactive application, it is difficult to precompute the correspondence information for every possible value of w. Fortunately, the correspondence information of contours has enough coherence such that we can store the same information over a range of isovalues. The main idea is to label every edge in CT^{t+1} with the correspondence information for the entire range of isovalues.

Once this correspondence information is computed as a preprocessing step, a graph showing the topology changes of time-varying isosurfaces can be immediately constructed for any selected isovalue. This graph is called the *Topology Change Graph (TCG)*. TCG is constructed by creating a node for every intersection point and connecting each pair of intersection points where their representative contours correspond to each other (Fig. 1c). We label TCG with additional quantitative information such as surface area and volume of each contour. Some of the applications of TCG are as follows:

- **Dynamic Structure Extraction**: One can determine the topology changes of time-varying isosurfaces (e.g., *merge*, *split*, *create*, *disappear*, and *genus change* of contours).
- **Feature Tracking**: One can interactively segment, track, quantify, and visualize the evolution of any individual contours.

The main contributions of this paper are: 1) define the temporal correspondence between contours, 2) create an algorithm to compute the correspondence information in time-varying contour trees, and 3) apply the correspondence information to fast construction of TCG for any selected isovalue.

The remainder of this paper is organized as follows: After reviewing related papers in Section 2, we define the temporal correspondence of contours and test conditions in Section 3. In Sections 4 and 5, we describe an algorithm to construct contour trees and compute correspondence information in time-dependent contour trees. Section 6 describes how to construct a topology change graph. In Sections 7 and 8, we compute geometric properties of contours in a contour tree and build an interactive system to segment and track the evolution of interesting contours, respectively. We present experimental results and real-life applications

of TCG for dynamic structure extraction and feature tracking in Section 9. Finally, in Section 10, we conclude this paper.

2 Related Work

Contour Tree

CT has been used in various fields such as image processing and GIS [23], [25]. Our main interest is using it in visualization. Van Kreveld et al. [27] described an $O(M \log M)$ and $O(M^2)$ algorithm to construct a contour tree from a 2D and 3D scalar field, respectively. The function is defined on a simplicial mesh with M elements and N vertices. Tarasov and Vyalyi [26] improved the time complexity to $O(M \log M)$ in the 3D case. Carr et al. [6] simplified Tarasov and Vyalyi's algorithm to construct the contour tree in all dimensions. The join tree and split tree are constructed and merged to build the contour tree in $O(M + N \log N)$. Pascucci [17] computed Betti numbers of contours to distinguish different topology of contours within an edge of CT. The divide and conquer approach [18] allows output-sensitive construction [8] of contour trees from a function defined on a rectilinear grid and easy extension to parallel implementation. Carr and Snoeyink [5] used CT as an interface to display topological structures of isosurfaces and segment individual contours in a scalar field. They computed path seeds for each edge, which generate a seed cell [1] necessary for rapid extraction of a selected contour in runtime. CT evolves as the function changes over time. Edelsbrunner et al. [10] combined the evolving sequence of contour trees defined from continuous space-time data into a single data structure. However, utilization of this data structure in visualization applications is not addressed. In contrast, in this paper, we focus on the development of interactive time-varying visualization and quantification tools, as described in Sections 6, 7, and 8, from the discretely evolving sequence of contour trees.

Isosurface Extraction in Time-Varying Fields

Visualization of time-varying fields has been a challenging problem because of overwhelming data sizes and heavy computation requirements. Time-based data structures [20], [24] are used to minimize unnecessary I/O access and support out-of-core isosurface extraction [7] in time-varying fields. The high-dimensional isosurfacing approaches [3], [28] consider time-dependent data in four-dimensional space. First, a three-dimensional solid mesh $\{(x, t)|f(x, t) = w\}$ is generated. Then, an isosurface at time t_0, $\{x|t(x) = t_0\}$, is extracted from the mesh.

Since the data sets are often large and contain many timesteps, it is useful to automatically detect significant timesteps and isovalues containing interesting features. The *contour spectrum* [2] computes and shows geometric and topological properties such as surface area, volume, and gradient integral of isosurfaces over all isovalues and timesteps. A similar interface, called *contour plane* [15], displays the number of contours over all isovalues and timesteps in a 2D plane.

Feature Tracking

Silver and Wang [21], [22] define a feature in a volume as a region of interest which satisfies a predefined thresholding criteria. After feature extraction, they perform a correspondence matching test of features based on the degree of overlap in order to track and quantify the evolution of each isolated feature. Dynamic events of the features are classified as continuation, creation, dissipation, bifurcation, and amalgamation, which can be depicted with a graph [19]. High-dimensional isosurfacing can be applied to feature tracking [14]. The correspondence relationship of contours can change only at the value of critical points in three-dimensional and four-dimensional functions. Based on the observation, the correspondence test is performed for each interval of adjacent critical values, which allows efficient isosurface tracking for every possible isovalue [13]. The process is accelerated by limiting the test to the

contour components that are associated with a critical point. Our method generates similar correspondence results, but we do not perform such explicit tests for checking contour correspondence. Our method also naturally allows us to track and precompute the degree of contour overlap as we visit each vertex in order. This is quite important for interactive construction of TCG because runtime overlap computation for large contours in every timestep would reduce the interactivity.

3 Contour Correspondence

Given two isosurfaces I^t and I^{t+1} from time t and $t+1$, a correspondence test determines whether a contour $C_k^t \in I^t$ corresponds (evolves) to a contour $C_{k'}^{t+1} \in I^{t+1}$. There are several ways to define the contour correspondence as it is difficult to know how an isosurface changes between the two sampled timesteps without making some specific assumption. We provide our own rule for the correspondence test.

A region defined as $X^t(w) = \{x|f^t(x) \geq w\}$ is termed an *object set*. An object set consists of connected components, called *objects* X_k^t. We can represent $X^t(w) = \{X_1^t, \ldots, X_n^t\}$. Similarly, we define $Y^t(w) = \{x|f^t(x) \leq w\} = \{Y_1^t, \ldots, Y_m^t\}$. We term X_k^t an *upper object* and Y_k^t a *lower object* for convenience. A contour always has only one upper object and one lower object which have the contour on the border. For example, green regions A and B in Fig. 2 are the upper object and lower object of C_1^t, respectively.

We define contour correspondence as follows:

Definition 1: Contour Correspondence

Consider two contours, C_k^t and $C_{k'}^{t+1}$. Suppose C_k^t is on the border of an upper object X_a^t and a lower object Y_b^t, and $C_{k'}^{t+1}$ is on the border of $X_{a'}^{t+1}$ and $Y_{b'}^{t+1}$.

There is overlap between upper objects X_a^t *and* $X_{a'}^{t+1}$ and *there is overlap between lower objects* Y_b^t *and* $Y_{b'}^{t+1}$.

$\Leftrightarrow$ C_k^t corresponds to $C_{k'}^{t+1}$, denoted as $C_{k'}^{t+1} \leftarrow C_k^t$ (i.e., C_k^t and $C_{k'}^{t+1}$ are the same contour at different timesteps).

Fig. 2 illustrates an example of the correspondence test. There are two contours, C_1^t and $C_{2'}^t$, at time t, and three contours, C_1^{t+1}, C_2^{t+1}, and C_3^{t+1}, at time $t+1$. Upper objects of C_1^{t+1} and C_2^{t+1} overlap an upper object of C_1^t and so do lower objects. On the other hand, the upper object of C_3^{t+1} does not overlap the upper object of C_1^t and C_2^t. The lower object of C_2^t does not overlap other lower objects. The correspondence relationships of the contours are shown in the red box of Fig. 2.

According to Definition 1, two large contours which have even a small overlapping region will correspond. In practice, this may not be desirable. We may add an additional condition for the degree of overlap in the definition such that the correspondence between contours requires *significant* overlap between upper/lower objects. The criteria of the *significant* overlap is determined as

$$\frac{V\left(X_a^t \cap X_{a'}^{t+1}\right)}{min\left(V\left(X_a^t\right), V\left(X_{a'}^{t+1}\right)\right)}, \frac{V\left(Y_b^t \cap Y_{b'}^{t+1}\right)}{min\left(V\left(Y_b^t\right), V\left(Y_{b'}^{t+1}\right)\right)} > \epsilon, \quad (1)$$

where ϵ is a threshold value for specifying the degree of the contour correspondence. $V(O)$ is a volume of an object O. $V(A \cap B)$ is a volume of an overlapping region between objects A and B. Fig. 3 provides an example for testing the degree of overlap.

Definition 1 with (1) can be thought of as an extension of feature correspondence [21] to contour correspondence in the sense that they are based on overlap between related objects. The definition is justified by the general observation that the same objects of successive timesteps have significant overlap, when time sampling is sufficient, and successful real applications [16], [21].

One of the canonical ways to define the contour correspondence is to construct a higher dimensional function $f^{t,t+1}$ interpolated between f^t and f^{t+1} and check whether C_k^t is continuously followed by $C_{k'}^{t+1}$ in $f^{t,t+1}$. However, this natural definition may also cause undesirable matchings in practice, as shown in Fig. 4. We adopt Definition 1 with (1) that gives flexibility in controlling the criteria for the degree of contour correspondence. The flexibility allows us to reduce the possibility of undesirable matchings.

4 Contour Trees

The Contour Tree (CT) with a vertex set V and an edge set E is defined from a scalar field f as follows: V consists of critical points of f where a contour is created, merged, split, and destroyed. We define a contour class as a maximal set of continuous contours which do not contain the critical points. An edge set E consists of edges connecting two critical points where a contour class is created and destroyed.

Our algorithm for computing correspondence information between two contour trees is based on [6]. This section provides high-level algorithm descriptions for constructing and merging the *Join Tree* (JT) and *Split Tree* (ST), which enable us to build CT. Since construction of JT and ST is symmetric, we only describe the algorithm for JT construction.

Starting from the maximum function value, we continuously decrease an isovalue w and incrementally mark regions $X(w) = \{x|f(x) \geq w\} = \{X_1, X_2, \ldots X_n\}$ on the domain where X_k is a connected component. Each connected component of the marked regions is conceptually the same as an upper object. As an isovalue passes through the function value of a local maximum, called an *upper leaf*, a new component is created. At this moment, a JT node for the upper leaf is created. In the case of ST construction, it is called a *lower leaf*. As an isovalue passes through a joining saddle point, called a *join*, two or more components are merged into one. A *split* is defined similarly to the *join*. A new JT node for the saddle point is created and edges connecting the new node and the node for the latest critical point which belongs to the same object are made. When w reaches the global minimum value, a JT node is created and connected to the node for latest critical point. In actual implementation [6], each vertex in a domain mesh is marked in decreasing order based on the function value, which causes the creation and, later, the coalescence of upper objects. When a vertex is marked, the vertex is connected to the lowest vertex of neighboring objects, which finally forms JT.

The upper leaves of JT and lower leaves of ST are successively deleted and adjacent edges of the leaves are inserted to form an Augmented Contour Tree (ACT). CT can be obtained by successively deleting regular vertices in ACT.

5 Correspondence Computation

In this section, we present our algorithm to compute the correspondence information of time-varying contours over all isovalues as a preprocessing step. Let e_k^t represent an edge of CT^t. The correspondence information is computed by labeling an edge e_k^{t+1} with a set of edges E^t within a function range $[f_a, f_b]$. This means a contour represented by an intersection point on e_k^{t+1} evolves from a set of contours represented by intersection points on $e_{k'}^t \in E^t$ for an isovalue $w_0 \in [f_a, f_b]$. The goal is to label every edge in CT^{t+1} for the entire function range.

Definition 1 and the process to construct join/split trees have an interesting relationship. The process of JT construction is similar to checking for coalescence among upper objects as an isovalue w decreases from the highest function value. Given f^t and f^{t+1} on the same domain, we start from the highest function value, gradually decrease the isovalue, and mark the regions where the function value is greater than the isovalue in f^t and f^{t+1} at the same time. The marked regions form upper objects from time t and $t + 1$, which are created and later overlap each other. When two upper objects from time t and $t + 1$, X_a^t and X_b^{t+1}, collide at a point x_c and isovalue w_0, X_a^t and X_b^{t+1} begin to have an overlapping area. Since the two upper objects grow as an isovalue decreases, they always overlap with each other after the collision. This relationship is exactly the same for ST construction and checking the overlap of two lower objects from time t and $t + 1$. We use join and split trees of f^t and f^{t+1} to compute the overlap information of upper and lower objects over all isovalues. Since Definition 1 requires the overlap of upper objects *and* lower objects, we must compute the intersection of the contour correspondence information in JT^{t+1} and ST^{t+1}. Note that we use the term *collision* and *overlap* for objects from different time steps and *merge* for objects from the same timestep during the growth of objects.

Fig. 5 shows the growth of upper objects in f^t and f^{t+1} on the same domain when w is continuously decreased from the maximum value. We use the same functions for Figs. 2,3,5, 6, and 9. Note that upper objects in time t and $t + 1$ are created and merged and collide with each other at certain isovalues. At such critical points, correspondence relationship of related objects are updated. Fig. 6 demonstrates that JT, ST, and CT capture such topological changes.

There are five steps for computing contour correspondence information in CT^{t+1} (Fig. 6). Note that the definition of contour correspondence is applied to the contour trees over continuous ranges of isovalues. A point on an edge of JT and CT represents an upper object and a contour, respectively. In the rest of this paper, we refer to "a contour/object represented by a point between an edge e and the line $y = w$ for an isovalue w" as "a contour/object with w on e" for brevity.

- **Step 1**: Assign an id to each edge in CT^t.
- **Step 2**: Label each edge e in JT^t/ST^t with a set of edges E of CT^t. An upper/lower object with w on e should contain the contours with w on $e' \in E$ on the border.
- **Step 3**: Label each edge e in JT^{t+1}/ST^{t+1} with a set of edges E of CT^t. An upper/lower object with w on e in time $t + 1$ should overlap upper/lower objects in time t that contain contours with w on $e' \in$ on the border.
- *Step 4*: Convert the labeled JT^{t+1}/ST^{t+1} into the form of a contour tree, $CT^{t+1}(JT)$ and $CT^{t+1}(ST)$. Consider that the upper/lower object represented by a point p' of $JT^{t+1}/$

ST^{t+1} contains a contour represented by a point p of the contour tree on the border. The label at p should be the same as the label at p'.

- **Step 5**: Label each edge e of CT^{t+1} with a set of edges E of CT^t such that a contour with w on e evolves from a set of contours with w on $e' \in E$. E is computed as the intersection of labels (i.e., edge sets) in $CT^{t+1}(JT)$ and $CT^{t+1}(ST)$.

Actual correspondence matching occurs in Step 3 because Definition 1 requires the comparison of upper/lower objects at time t and $t + 1$. Steps 1, 2, 4, and 5 convert the label information from CT to JT/ST or from JT/ST to CT.

Since the construction of JT and ST in Step 1 through Step 5 is repetitive, we omit the case of ST in the algorithm description. Overall algorithms for the five steps are described in this section. Algorithm pseudocode is provided in the Appendix.

Step 1 is performed by the algorithm used for CT construction described in Section 4. Step 2 processes each vertex v of the domain mesh M^t in decreasing order based on its function value. It can be thought of as traversing and labeling JT in time t from the upper leaves to a global minimum as an isovalue, w, decreases from the highest function value. The label changes only at the critical point. If v is a critical point, a node for JT^t is created and connected with its parent nodes, which belong to the same objects with v. The label is updated in an incremental way. An edge id for the destroyed contour class(es) is deleted from and an edge id for the newly created contour class(es) is inserted into the label (i.e., edge set) which is kept in the parent vertex of v in JT^t. The blue boxes of Fig. 6 provide an example. Since the contour class e_1 is destroyed and e_2 and e_3, which belong to the same upper object, are created at $w = f^t(v_1)$, the label $\{e_1\}$ is updated into $\{e_2, e_3\}$ at v_1 in JT^t. In the same way, since e_2 is destroyed at $w = f^t(v_2)$, the label $\{e_2, e_3\}$ is updated into $\{e_3\}$ at v_2.

Step 3 processes each vertex v of domain meshes M^t and M^{t+1} in decreasing order based on the function value of v. We assume the vertex positions and connectivity of domain meshes are the same for each timestep. The regions $\{x|f^t(x) \geq f^tv(v)\}$ and $\{x|f^{t+1}(x) \geq f^tv(v)\}$ are marked incrementally on M^t and M^{t+1} as each vertex v is processed. t_v is the timestep to which the vertex v belongs. Recall that the marked region is conceptually the same as an *upper object*. Upper objects in time t and $t + 1$ grow for each iteration of the vertex processing loop. It can also be thought of as traversing the join trees and detecting overlap of upper objects in time t and $t + 1$ as w decreases, like in Step 2. During a vertex processing loop, there can be three kinds of events: 1) Two upper objects collide, one object from time t and the other from time $t + 1$, 2) contour topology changes in time $t + 1$, and 3) contour topology changes in time t. Events 2) and 3) cannot happen at v at the same time. Each case is explained as follows.

As we process each vertex v in decreasing order, upper objects increase in size. We need to check whether two objects, X_k^t and $X_{k'}^{t+1}$, collide at a point x_c during the growth. Two objects that have already collided are not considered for the test again. Because we use simplicial domain meshes and PL interpolation, x_c is always placed on an edge of the mesh. To detect the collision, we check the neighboring vertices of v for each iteration of the vertex processing loop. If a neighboring vertex v' is covered by and v is not covered by an upper object from the other timestep of v, the collision point x_c on the edge (v, v') and $f(x_c)$ is computed using the function values at v and v' in time t and $t + 1$ (see Fig. 7a). The label in JT^{t+1} is updated when w passes through $f(x_c)$ if the two objects which meet at x_c have not collided yet.

The overlap of the two upper objects indicates that the contours on the border of the two objects correspond to each other. Therefore, a new node having $f(x_c)$ in JT^{t+1} is created and connected with a parent node. The label at the collision point (red nodes in Fig. 6) is updated into the union of labels (i.e., edge sets) of the two colliding objects. We maintain a table indicating

whether any combination of two upper objects from time t and $t + 1$ overlap or not during Step 3 because we have to frequently check whether any two upper objects overlap. In Fig. 6b, marked with the dashed red arrow, collision occurs between green and blue upper objects. The label (i.e., edge set), $\varnothing$, on the point of JT^{t+1} is updated into the union of the edge sets, $\{e_1\} \cap \varnothing$, on the points in JT^t and JT^{t+1}.

In order to specify the additional condition (1) for the degree of overlap in contour correspondence definition, we need to detect when the two collided upper objects *significantly* overlap as specified by a predefined threshold value instead of detecting the collision point x_c. For the detection, each object X_k^{t+1} in time $t + 1$ maintains a list of objects in time t which have collided with X_k^{t+1} and the number of vertices in the overlapping region. Each object in time t and $t + 1$ maintains the number of vertices which belong to the object, too. These numbers of vertices approximate the volume of each object and an overlap region. It is sufficient to measure the volume only at the value of each vertex because there cannot be a vertex that has a value between any two adjacent data values and, thus, the degree of overlap does not change between the two adjacent values. In each loop of vertex processing, we update the numbers of the related objects and check whether two objects which have collided start to have (or lose) significant overlap. When they start to have (or lose) significant overlap, we perform the same process for updating the label as described in the previous paragraph.

If v is an *upper leaf* at time $t + 1$, a new node n having $f^{t+1}(v)$ is created. If there is already an upper object X_k^t at time t in the place of v, n is labeled with the same edge set as X_k^t. Otherwise, the label of n becomes an empty set. Also, if v is a *join* at $t + 1$, a new node n having $f^{t+1}(v)$ is created. Since two or more objects are merged at v, n is labeled with the union of edge sets for the merged objects and connected to the parent nodes.

If v is an *upper leaf* at time t, the CT^t edge id of the newly created contour is inserted into the edge set of the object at time $t + 1$, which contains the point v. If v is a *join*, *split*, or *lower leaf* at time t, then find objects X_k^{t+1} at time $t + 1$ that overlap the object at time t, which v belongs to, and update the edge set of X_k^{t+1}. For other cases, a node's edge set is maintained until updated because of the above-listed events. In Fig. 6, the topology of contours in time t changes at the levels of green dashed arrows. Between Fig. 6b and Fig. 6c, a contour on e_1 of the green object is split into two contours on e_2 and e_3. The blue object overlapping the green object updates its label by subtracting $\{e_1\}$ and adding $\{e_2, e_3\}$. This process is the same for other green dashed arrows. At the join node of JT^{t+1}, the edge set is updated into the union of the edge sets in the parent nodes.

Since one upper object may contain several contours on its border, an edge in JT^{t+1} corresponds to a set of edges in CT^{t+1}. Using this property, the fourth step visits each edge of JT^{t+1} and copies its label to the corresponding edges in CT^{t+1}. For example, in Fig. 7b, an edge in JT^{t+1} corresponds to two edges e_1^{t+1} and e_2^{t+1} of CT^{t+1}. Each label on this edge computed from Step 3 is copied to e_1^{t+1} and e_2^{t+1}.

After Step 4, two contour trees with different labels computed from JT^{t+1} and ST^{t+1} are generated. The fifth step visits each edge of CT^{t+1} and computes the intersection of edge sets (i.e., labels) in the two contour trees.

Time Complexity Analysis

N, M, and c^t are the number of vertices, tetrahedra, and critical points in f^t, respectively. The number of upper or lower objects at a certain isovalue in f^t cannot be greater than c^t. The entire algorithm consists of five steps.

- Step 1: $O(N \log N + M)$. This is analyzed in [6], [18].
- Step 2: $O(N \log N + (c^t)^2)$.
- Step 3: $O(N \log N + M + (c^t)^2 \cdot c^{t+1})$.
- Step 4: $O((c^{t+1})^2 c^t)$.
- Step 5: $O((c^{t+1})^2 (c^t)^2)$.

$N \log N$ is for sorting vertices and M is for visiting adjacent vertices for each vertex. In Step 2, updating a label when processing a critical point may take $O(c^t)$, which makes the total complexity $O(N \log N + (c^t)^2)$. In Step 3, collision can occur no more than $c^t \cdot c^{t+1}$ times because there can be at most c^t and c^{t+1} objects at time t and $t + 1$, respectively. For each collision, we need to compute the union of edge sets with $O(c^t)$ time. The complexities for other parts of Step 3 are minor. The total cost for Step 3 is $O(N \log N + M + (c^t)^2 \cdot c^{t+1})$. In Step 4, visiting each edge e of JT^{t+1} takes $O(c^{t+1})$. For each e, there can be at most c^{t+1} edges of CT^{t+1} corresponding to e. Also, there can be at most c^t labels in e. Therefore, Step 4 takes $O((c^{t+1})^2 c^t)$. In Step 5, we take intersections of edges and their edge sets in the contour trees computed from JT^{t+1} and ST^{t+1}. As a result of Step 4, $CT^{t+1}(JT)$, $CT^{t+1}(ST)$ has at most $(c^{t+1})^2 c^t$ edges. For each intersected edge, we need to percorm edge set (i.e., label) intersection which takes $O(c^t)$. The comst for Step 5 is $O((c^{t+1})^2 (c^t)^2)$.

Processing Steps 4 and 5 is generally slow (see the timing results in Section 9). Although a labeled CT^{t+1} is an ideal structure for capturing the temporal correspondence information, all the information in the labels of CT^{t+1} is embedded in the labels of JT^{t+1} and ST^{t+1}, which are the output of Step 3. In runtime, every point p^{t+1} in CT^{t+1} can be mapped to corresponding points p^t_{JT} and p^t_{ST} on JT^{t+1} and ST^{t+1}, respectively, because a contour is contained by one upper object and one lower object. The edge set (i.e., label) on p^{t+1} of CT^{t+1} can be computed by intersecting the edge sets on p^t_{JT} and p^t_{ST}. In practice, we perform only Steps 1, 2, and 3 to save preprocessing time and maintain labeled JT^{t+1} and ST^{t+1} for runtime correspondence queries.

6 Topology Change Graph

This section classifies topological events of time-varying isosurfaces and describes a runtime algorithm to construct a graph representing the topological events. When an isovalue w is selected, isosurfaces for each time t are defined as $I^t = \{C^t_1, \ldots, C^t_n\}$. Six topological events are defined based on contour correspondence as follows:

- **create:** C^t_k is created in time t if no contour in time $t - 1$ evolves to C^t_k.
- **disappear:** C^t_k disappears in time $t + 1$ if no contour in time $t + 1$ evolves from C^t_k.
- **merge:** More than one contour in time $t - 1$ is merged to form C^t_k at time t if they evlove to C^t_k.
- **split:** C^t_k is split in time $t + 1$ if more than one contour in time $t + 1$ evolves from C^t_k.
- **continue:** C^t_k continues in time $t + 1$ if only one contour in time $t + 1$ evolves from C^t_k.
- **genus change:** C^t_k changes its genus in time $t + 1$ if Betti numbers of a contour in time $t + 1$ which evolves from C^t_k are different from that of C^t_k. Betti numbers can be precomputed for all contours [18].

The challenge in topology tracking of contours over time is to find a correspondence between the contours of consecutive isosurfaces I^t and I^{t+1} for all t in the time sequence. Using the algorithms described in Section 5, we construct the contour trees CT^t and CT^{t+1}. When we select an isovalue w_0 in runtime, we obtain the intersection points between CT^t and CT^{t+1} and a line $y = w_0$. Each intersection point in CT^{t+1} represents a contour C_k^{t+1}. Since correspondence information is already computed for every edge of CT^{t+1}, a point of CT^{t+1} representing C_k^{t+1} has an edge set E (i.e., label). Therefore, C_k^{t+1} evolves from a set of contours with w_0 on $e \in E$. The *genus change* of a contour is detected by comparing the Betti number of correlated contours. The genus change event may occur together with other events for a contour.

All the above topological events can be visualized via a graph, called the *Topology Change Graph* (TCG). Each contour C_k^t at time t is represented as a node N_k^t and a set of such nodes S^t at time t are placed vertically. A sequence of nodes $S^1, \ldots, S^T$ are placed horizontally in time order (Fig. 8b). If C_k^t evolves to $C_{k'}^{t+1}$, two nodes $N_{k'}^{t+1}$ and N_k^t are connected with an edge. This process is performed for all contours in each timestep sequentially to construct TCG. The Betti numbers are stored in each node as a property and are used to detect the genus change of a contour. Typically, Betti number 1 is of a particular interest as it indicates the number of independent tunnels in a closed contour manifold.

Fig. 8 demonstrates TCG construction from labeled contour trees. When an isovalue w is selected in runtime, two points in time t and three points in time $t + 1$ are identified. The points p_1 and p_3 have an edge set $\{e_3\}$ and are mapped to the point on e_3 in CT^t. The label on p_2^{t+1} is an empty set; p_2^{t+1} is mapped to no point in CT^t. The mapping of the points constructs the TCG.

7 Time-Varying Quantification

In this section, we quantify the geometric features of contour evolution. The geometric properties such as surface area and volume for each contour are computed and labeled in the topology change graph. This quantitative information helps users find dynamic features of contour evolution and isolate and track interesting contours. For example, the nodes of the graph can be colored based on the surface area or volume quantities. Users can guess which components are significant and how a specific component evolves over time by looking at the quantity changes. In many cases, contours with small surface area/volume are considered insignificant noise.

Conventional algorithms [21] extract features (e.g., contour surface) first and then compute the geometric properties of the features. However, such approaches are not suitable for interactive applications because surface extraction is an expensive process. Our approach efficiently precomputes the geometric properties of all possible contours for each timestep. This allows accurate and real-time evaluation of the properties for any selected contour.

Consider a 3D scalar field f defined on a simplicial mesh. The area and volume of isosurfaces in an ith simplex can be represented with a univariate B-spline function $A_i(w)$ and $V_i(w)$, respectively. When such functions for all simplices are merged, two single B-spline functions, $A(w) = \Sigma A_i(w)$ and $V(w) = \Sigma V_i(w)$, are constructed. If a user selects an isovalue w_0 as an interactive parameter, the area and volume of $I(w_0)$ is evaluated from the functions. We refer to [2] for a detailed description of $A(w)$ and $V(w)$ computation.

We can compute the area and volume of every possible contour in a similar manner by using a contour tree. A class of continuous contours represented by an edge e covers a region (e.g., blue region in Fig. 9b). Let's assume a set of simplices $S_e = \{s_{(e,1)}, \ldots, s_{(e,n)}\}$ covers the region.

An area and volume function of a contour with w on e, $A(e, w)$ and $V(e, w)$, is the sum of a B-spline function for each simplex $s_{(e,k)}$.

$$A(e,w) = \sum_{k=1}^{n} A_{s_{(e,k)}}(w).$$

Volume computation of a contour involves the entire inside or outside region in addition to the region covered by a contour class of e. The inside and outside regions are identified by a set of upper edges or lower edges of e. An edge e separates a contour tree into a set of upper edges and a set of lower edges. See Figs. 9a and 9b.

$$V(e,w) = \sum_{k=1}^{n} V_{s_{(e,k)}}(w) + \sum_{e' \in E(e)} V_{e'}.$$

$E(e)$ is a set of upper edges or a set of lower edges of e, depending on the user's choice on which side to take. $V_{e'}$ is a volume of the region covered by a contour class of e'. $V_{e'}$ is computed for each edge $e' \in E(e)$ and is used to compute $\Sigma_{e' \in E(e)} V_{e'}$ in an incremental way through the hierarchical structure of a contour tree.

Once $A(e, w)$ and $V(e, w)$ are precomputed for each edge e of a contour tree, the area and volume of any contour with w_0 on e are quickly evaluated from $A(e, w_0)$ and $V(e, w_0)$ in runtime (Fig. 9c). Each node in TCG can be colored based on the geometric quantity of the contour the node represents (Fig. 10c).

An algorithm to compute S_e is as follows: Let $e = (v_1, v_2)$ such that $f(v_1) \leq f(v_2)$. First, we pick any seed cell c in e and use a propagation method similar to [1]. This takes $O(|S_e|)$.

Input: *CT*, path seeds [5], e

Output: S_e

1. $c \leftarrow$ a seed cell in an edge e of *CT*;
2. Enqueue c;
3. Visit(c);
4. **while** queue is not empty **do**
5. $s \leftarrow$ Dequeue();
6. $t \leftarrow$ GetFaces(s);
7. **for** each face t_i of s, $i = 1, 2, 3, 4$
8. **if** Min(t_i) < $f(v_2)$ and Max (t_i) > $f(v_1)$ **then**
9. $c \leftarrow$ tetrahedron sharing the face t_i with s;
10. **if** c is not visited **then**
11. Enqueue(c);
12. Visit(c);

In the above pseudocode, GetFaces(s) returns the four triangles of a tetrahedron s. Min(t) and Max(t) return the minimum and maximum function values defined in the triangle t. Line 8 in

the code checks whether the triangle t_i overlaps with the continuous contour class of the edge e. Visit(c) inserts the cell c into S_e.

8 Interactive Contour Tracking

When time-varying isosurfaces have many evolving contours including noise, users may wish to segment and visualize a subset of contours. This allows the user to focus on the evolution of interesting features. The topology change graph combined with quantitative information is used as an interface to help identify significant contours and their dynamic patterns. In this section, we describe an interactive algorithm to select and extract specific contours and track their evolution over time.

We use a seed set propagation method [1] for contour surface extraction. A seed set S is defined as a subset of all cells, with the property that any isosurface component intersects with at least one cell in S. One advantage to using the seed set propagation method is its ability to segment a single connected component of an isosurface. Since we use a simplicial mesh and PL function, there can be at most one piece of an isosurface in a cell. Starting from a contour fragment in a seed cell, the propagation method incrementally traces and triangulates the cells that contain the remainder of the contour. Propagating a contour for each seed cell constructs a complete isosurface. This process is efficient because we can avoid visiting unnecessary cells which is the main bottleneck of isosurface extraction.

CT is useful for generating a minimal seed set and isolating individual contours. When a user selects a point on an edge of CT, a seed cell that intersects with a contour represented by the point is computed in runtime. The seed cell is used for efficient extraction of the contour [5].

Consider TCG for an isovalue w_0. When a user selects a node N_k^t at time t in the graph, the point in CT^t corresponding to N_k^t is identified. A seed cell for this point is computed and used for contour extraction. Edge connectivity of the graph is used for tracking and displaying the evolution of a selected contour. Contours in time $t + 1$ represented by the nodes connected to N_k^t can be quickly extracted in the same way as the extraction of C_k^t. Tracking backward can be done in the same manner. Fig. 10 depicts this process.

9 Results

We tested two time-varying data sets generated from hemoglobin dynamics and turbulent vortex simulations. The data sets are defined on rectilinear grids. We divided each cubic cell into six tetrahedra because the algorithm requires simplicial meshes. This cell decomposition reduces the speed of the program and generates a few undesirable artifacts [4] in the extracted surface. Visit our Website [11] for additional details, including images, videos, and descriptions.

The first time-dependent data set is an approximate electron density map of a deforming hemoglobin molecule (Fig. 11a). Goodsell's Website [12] describes the hemoglobin molecule and its dynamics. Hemoglobin is a protein that binds to oxygen in oxygen-rich areas (lung) and releases the oxygen in oxygen-poor areas (tissues). The hemoglobin dynamics data set represents this *oxy-deoxy* process and has 30 timesteps with 128^3 sized electron density map for each timestep. As shown in Fig. 11, the contour tree (Fig. 11b) of the density field (Fig. 11a) at time 1 indicates that the hemoglobin molecule consists of four (two sets of identical) polypeptide chains. A contour component of each chain is constructed and visualized with a different color. Each chain has a flat ring structure called a *heme*, which is the active site in the oxy-deoxy process (Fig. 11e). The rest of the polypeptide chain is called a globin. The contour tree (Fig. 11b) shows the contour for each chain is divided into three components: a heme

(ring), iron, and globin. Using the topology change graph, we can detect when and where the oxygen is bound to and released from the heme group, shown in Fig. 11d. We can also track, quantify, and visualize the evolution of each heme group (Figs. 11e, 11f, and 11g).

The other data set is a pseudospectral simulation of coherent turbulent vortex structures [21] with a 128^3 resolution and 33 time steps. Fig. 10 shows the result of contour segmentation and tracking. When an isovalue 6.5 is selected, a TCG is constructed (Fig. 10c) where each node is colored with the surface area of its contour. A contour can be segmented and interactively tracked over time using TCG. The connectivity and colors of nodes are used to detect topological and geometric changes of contours.

We measured the time for computing correspondence information for two functions at timestep 1 and 2 on an SGI ONYX 2 system with R12000 processors and 25GB main memory. The timing results are summarized in Table 1. The computation for each sequence of functions (f^t, f^{t+1}) can be done independently in a parallel system. The result shows Steps 4 and 5 are computationally expensive. As we mentioned in the time complexity analysis of Section 5, all of the information in CT^{t+1} is embedded in JT^{t+1}/ST^{t+1}, which is the output of Step 3. Therefore, in practice, we can perform only Steps 1, 2, and 3 to label JT^{t+1}/ST^{t+1} for preprocessing. In runtime, when the correspondence information of a contour is requested, we can compute it from the labels of JT^{t+1}/ST^{t+1}.

Every runtime operation is performed at interactive rates. The construction of a topology change graph, quantification and tracking is completed in less than 0.1 sec for both data sets. Each of the three contour surfaces in Fig. 10d is extracted in 0.28s, 0.28s, and 0.29s, where the number of triangles are 12,025, 12,652, and 12,253, respectively. Generally, the contour extraction time increases linearly with the number of contour triangles.

10 Conclusion

We described an algorithm to compute correspondence information in time-dependent contour trees. We used this information to extend the benefits of a contour tree to interactive and quantitative visualization of time-varying scalar fields. First, we extracted a graph capturing the topology changes of time-varying isosurfaces. Second, we segmented, tracked, visualized, and quantified the evolution of user selected contours. Finally, we accelerated the extraction of a contour surface by generating the seed cells from each contour tree. We implemented an interactive user interface which adopts the above three features and allows users to detect significant topological and geometric changes of time-varying isosurfaces.

Acknowledgments

The authors are grateful to A. Thane for developing the CVC volume rendering tool (Volume Rover), K. Clarridge for her help with the presentation, Dr. A. Shamir for helpful discussions, Dr. D. Goodsell for providing the hemoglobin dynamics data set, and Dr. V. Fernandez, S.Y. Chen, and Dr. Silver for providing the vortex data set. This work was supported in part by US National Science Foundation grants ITR-ACI-022003 and ITR-EIA-0325550 and the US National Institute of Health grants P20 RR020647-01 and R01-GM074258-02.

Appendix

We provide pseudocodes for Steps 2, 3, and 4 in this section. The functions “CreateNode” and “Connect” generate the nodes and edges of output trees. The node is stored with its function value. The output trees are labeled with appropriate edge sets, ESET, in time t and $t + 1$. The ESET of an edge in an output tree is determined when one of the two nodes in the edge is created. $E(v_1, v_2)$ returns a CT edge id for the location of (v_1, v_2). $P(v)$ and $C(v)$ return the parent and child vertex connected to v in Augmented Contour Tree (ACT) [6]. Note that $P(v)$ or C

(v) may return more than one vertex if v is a *join* or a *split*. In such a case, the function "P" or "C" is applied multiple times. We use an array $N^t[v]$ to store the latest node in a join tree, which belongs to the same object with v I time t. ACT^t is constructed during the contour tree construction in Step 1.

TABLE 1
Timing Results for Correspondence Computation (Unit: Sec)

Data set	Step 1	Step 2	Step 3	Step 4	Step 5
Hemoglobin	116	13	357	5241	25118
Vortex	244	9	411	3683	3412

In Step 2, the vertices of the domain mesh M^t are sorted in increasing order and stored in an array *va*.

STEP 2

Input: ACT^t, *va*

Output: JT^t_E—labeled JT^t

1. **for** $i \leftarrow nv - 1$ **to** 0 // decreasing order
2. $v \leftarrow va[i]$;
3. **if** (v is *upper leaf*) **then**
4. $n \leftarrow$ CreateNode($f^t(v)$, $\{E(v, C(v))\}$);
5. **if** (v is *join* or *split*) **then**
6. $n \leftarrow$ CreateNode($f^t(v)$,ESET($N^t[\mathrm{P}(v)]$)−$\{E(v), P(v))\}$ + $\{E(v, C(v))\}$);
7. Connect(n, $N^t[P(v)]$);
8. **if** (v is *lower leaf*) **then**
9. $n \leftarrow$ CreateNode($f^t(v)$, ESET($N^t[\mathrm{P}(v)]$)−$\{E(v, P(v))\}$);
10. Connect(n, $N^t[P(v)]$);
11. **if** (v is *regular*) **then**
12. $N^t[v \leftarrow N^t[\mathrm{P}(v)]$;
13. **else** $N^t[v \leftarrow n$;

In Step 3, the vertices of M^t and M^{t+1} are sorted in increasing order and stored in an array *va*2.

STEP 3

Input: JT^t_E, ACT^t, ACT^{t+1}, *va*2

Output: JT^{t+1}_C—labeled JT^{t+1}

1. **for** $i = 2vn - 1$ **to** 0 // vertex processing in decreasing order
2. (v, *time*) $\leftarrow$ *va*2[i]; // *time* = t or $t + 1$. // collision occurs

3. **if** collisions between object sets X^t and X^{t+1} occur when w is decreased from $f(va2[i-1])$ to $f(va2[i])$ **then**
4. **for** each collision point x_c between objects X_k^t and $X_{k'}^{t+1}$
5. $lv_1 \leftarrow \text{LowestVtx}(X_k^t)$;
6. $lv_2 \leftarrow \text{LowestVtx}(X_{k'}^{t+1})$;
7. $n \leftarrow \text{CreateNode}(f(x_c), \text{ESET}(N^t[lv_1])+\text{ESET}(N^{t+1}[lv_2]))$;
8. $\text{Connect}(n, N^{t+1}[lv_2])$;
9. $N^{t+1}[lv_2] \leftarrow n$; // contour topology changes in time $t + 1$
10. **if** *time* = $t + 1$ **then**
11. **if** (v is *upper leaf*) **then**
12. $n \leftarrow \text{CreateNode}(f^{t+1}(v), \text{ESET}(N^t[\text{LowestVtx}(\text{V2X}(v, t))])$;
13. $N^{t+1}[v] \leftarrow n$;
14. **if** (v is *join*) **then**
15. $n \leftarrow \text{CreateNode}(f^{t+1}(v),\text{ESET}(N^{t+1}[\text{P}(v)]))$;
16. $\text{Connect}(n, N^{t+1}[\text{P}(v])$;
17. $N^{t+1}[v] \leftarrow n$;
18. *if* (v is *split* or *lower leaf* or *regular*) **then**
19. **if** (v_2 is the lowest of the whole tree) **then**
20. $n = \text{CreateNode}(f^{t+1}(v),\text{NULL})$;
21. $\text{Connect}(n; N^{t+1}[\text{P}(v)])$;
22. **else** $N^{t+1}[v] \leftarrow n$; // contour topology changes in time t
23. **if** *time* = t **then**
24. **if** (v = is *upper leaf*) **then**
25. **if** $\text{V2X}(v, t + 1) \neq \text{NULL}$ **then**
26. $n' \leftarrow N^{t+1}[\text{LowestVtx}(\text{V2X}(v, t + 1))]$;
27. $n = \text{CreateNode}(f^t(v), \{E(v, \text{C}(v))\}+\text{ESET}(n')$;
28. $\text{Connect}(n, n')$;
29. $N^{t+1}[\text{LowestVtx}(\text{V2X}(v, t + 1))] \leftarrow n$;
30. **if** (v is *join* or *split* or *lower leaf*) **then**
31. **for** each object $X_{k'}^{t+1}$ in time $t + 1$ which overlaps the object, $\text{V2X}(v, t)$, v belongs to
32. $w \leftarrow \text{LowestVtx}(X_{k'}^{t+1})$;
33. $n = \text{CreateNode}(f^t(v), \text{ESET}(N^{t+1}[w]) - \{\text{E}(v,\text{P}(v))\} + \{\text{E}(v,\text{C}(v))\})$;
34. $\text{Connect}(n, N^{t+1}[w])$;
35. $N^{t+1}[w] \leftarrow n$;

LowestVtx(X_k^t) returns the vertex with the lowest value which belongs to an upper object X_k^t. V2X(v, t_0) returns the upper object at time t_0 which covers the place of v. t_0 is either t or $t + 1$. If no upper object at time t_0 covers the place of v, NULL is returned. We use a Union-Find data structure [9] in order to maintain object membership for each vertex v during above procedures. The data structure is used for efficient implementation of the functions LowestVtx and V2X [6].

The input of Step 4 is two join trees labeled differently. We may apply Steps 1 and 2 to label a join tree, JT_E^{t+1}, in time $t + 1$. Let the output join tree generated from Step 3 be JT_C^{t+1}.

STEP 4

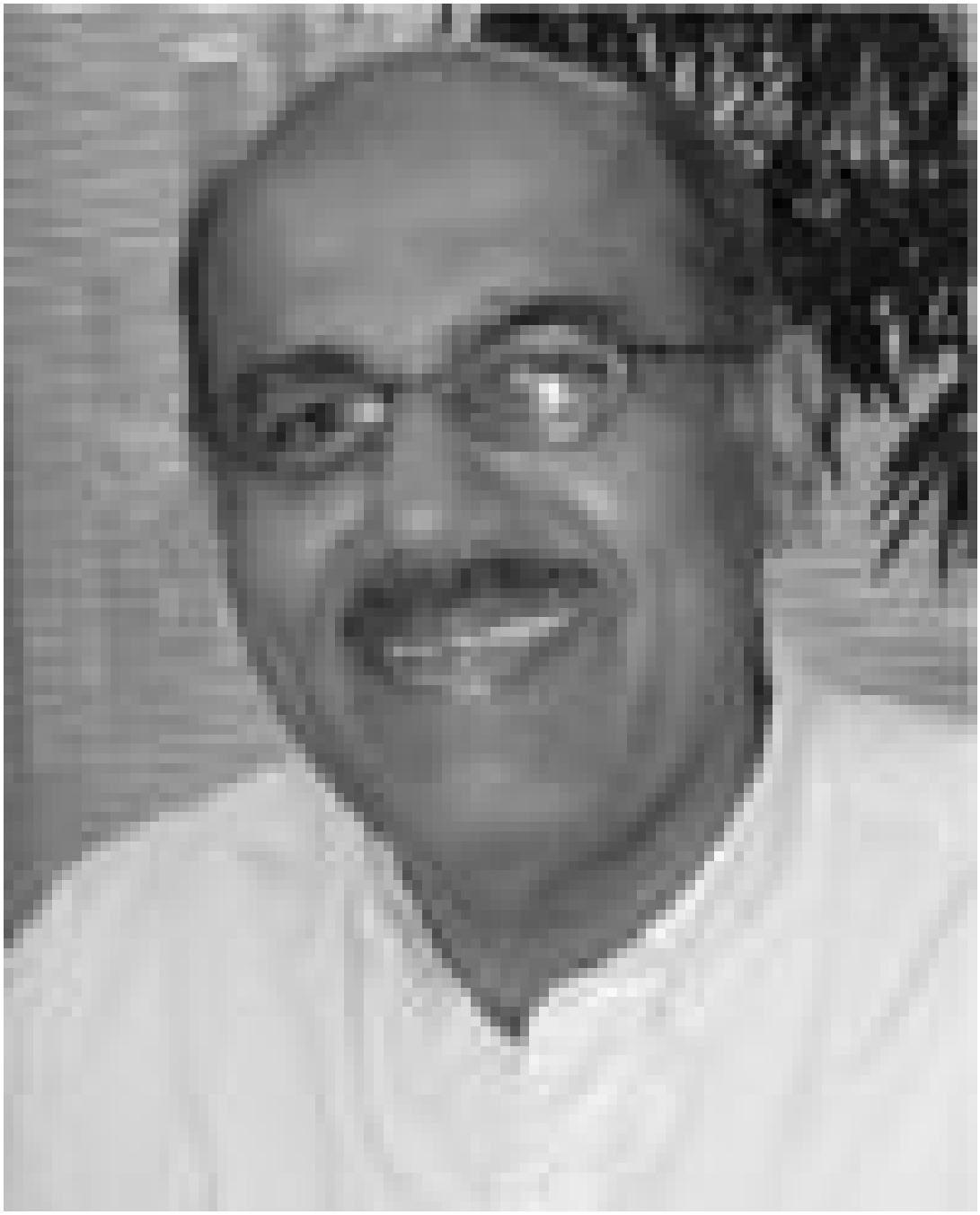
Input: JT_C^{t+1}, JT_E^{t+1}

Output: $CT^{t+1}(JT)$

1. $ea \leftarrow$ sorted edges of JT_E^{t+1}
2. **for** i = sizeof(ea) – 1 **to** 0
3. $(n_a, n_b) \leftarrow$ two nodes of an edge $ea[i]$;
4. Decompose $(n_a, n_b) \rightarrow (n_a, n_1), (n_1, n_2), \ldots, (n_m, n_b)$ based on the labels of JT_C^{t+1}; (see Fig. 7b)
5. **for** each $e_k \in$ ESET($ea[i]$)
6. Copy labels of $(n_a, n_1), \ldots, (n_m, n_b)$ to e_k. The labels are taken from JT_C^{t+1}.

Biographies

Bong-Soo Sohn is currently a PhD candidiate in computer sciences at the University of Texas at Austin. He received the BS degree in computer science from Seoul National University, Seoul, Korea, and the MS degree in computer sciences from the University of Texas at Austin in 1999 and in 2001, respectively. His research interests are in the areas of 3D computer graphics, visualization, and multimedia processing.



Chandrajit Bajaj graduated from the Indian Institute of Technology, Delhi, with the bachelor's degree in electrical engineering in 1980 and received the MS and PhD degrees in computer sciences from Cornell University in 1983 and 1984, respectively. He is the CAM Chair in Visualization Professor of Computer Sciences at the University of Texas at Austin, as well as the director of the Center for Computational Visualization in the Institute of Computational Engineering and Sciences (ICES). Prior to joining the University of Texas, he was a professor of computer sciences at Purdue University and director of the Purdue Center for Image Analysis and Visualization. Bajaj's research spans the areas of image processing, computer graphics, geometric modeling, visualization, and computational mathematics. Current research problems include denoising, reconstruction, and compression algorithms for volumetric and time-

dependent imaging, as well as data structures that support multiresolution finite element approximations of very large geometries and multiple function fields. He is also involved in integrated approaches to computational modeling, simulations, mathematical analysis, and interrogative visualization, especially for dynamic biomedical phenomena. He has more than 200 publications, has written one book, and edited three other books in his area of expertise. He is on the editorial boards for the International Journal of Computational Geometry and Applications, the *ACM Transactions on Graphics*, and *ACM Computing Surveys*. He is on numerous national and international conference committees and has served as a scientific consultant to industry. His research has been supported by DOE, NASA, the US National Science Foundation, NIH, and the Whitaker Foundation.

References

1. Bajaj CL, Pascucci V, Schikore DR. Fast Isocontouring for Improved Interactivity. Proc. 1996 Symp. Volume Visualization 1996:39–46.
2. Bajaj CL, Pascucci V, Schikore DR. The Contour Spectrum. Proc. IEEE Visualization Conf 1997:167–173.
3. Bhaniramka P, Wenger R, Crawfis R. Isosurfacing in Higher Dimensions. Proc. IEEE Visualization 2000 2000:267–274.
4. Carr H, Möller T, Snoeyink J. Simplicial Subdivisions and Sampling Artifacts. Proc. IEEE Visualization Conf 2001:99–108.
5. Carr H, Snoeyink J. Path Seeds and Flexible Isosurfaces Using Topology for Exploratory Visualization. Proc. IEEE TCVG Symp. Visualization (VisSym) 2003:49–58.
6. Carr H, Snoeyink J, Axen U. Computing Contour Trees in All Dimensions. Computational Geometry: Theory and Applications 2003;24(2):75–94.
7. Chiang Y-J. Out-of-Core Isosurface Extraction of Time-Varying Fields over Irregular Grids. Proc. IEEE Visualization Conf 2003:217–224.
8. Chiang Y-J, Lenz T, Lu X, Rote G. Simple and Optimal Output-Sensitive Construction of Contour Trees Using Monotone Paths. Computational Geometry: Theory and Applications 2003;30(2):165–196.
9. Cormen, TH.; Leiserson, CE.; Rivest, RL. Introduction to Algorithms. Mass.: MIT Press; Cambridge: 1990.
10. Edelsbrunner H, Harer J, Mascarenhas A, Pascucci V. Time-Varying Reeb Graphs for Continuous Space-Time Data. Proc. ACM Symp. Computational Geometry 2004:366–372.
11. 2005. Website for Time-Varying Contour Topologyhttp://www.ices.utexas.edu/bongbong/time_analysis
12. Goodsell, D. Hemoglobin: Cooperation Makes It Easier. 2005. http://www.scripps.edu/pub/goodsell/pdb/pdb41/pdb41_2.html
13. Ji G, Shen H-W. Efficient Isosurface Tracking Using Precomputed Correspondence Table. Proc. Eurographics-IEEE TCVG Symp. Visualization 2004:283–292.
14. Ji G, Shen H-W, Wenger R. Volume Tracking Using Higher Dimensional Isocontouring. Proc. IEEE Visualization Conf 2003:209–216.
15. Kettner L, Rossignac J, Snoeyink J. The Safari Interface for Visualizing Time-Dependent Volume Data Using Iso-Surfaces and Contour Spectra. Computational Geometry: Theory and Applications 2003;25(12):97–116.
16. Koegler WS. Case Study: Application of Feature Tracking to Analysis of Autoignition Simulation Data. Proc. IEEE Visualization Conf 2001:461–464.
17. Pascucci V. On the Topology of the Level Sets of a Scalar Field. Proc. 12th Canadian Conf. Computational Geometry 2001:141–144.
18. Pascucci V, Cole-McLaughlin K. Efficient Computation of the Topology of Level Sets. Proc. IEEE Visualization Conf 2002:187–194.
19. Samtaney R, Silver D, Zabusky N, Cao J. Visualizing Features and Tracking Their Evolution. Computer July;1994 :20–27.

20. Shen H-W. Isosurface Extraction in Time-Varying Fields Using a Temporal Hierarchical Index Tree. Proc. IEEE Visualization Conf 1998:159–166.

21. Silver D, Wang X. Tracking and Visualization Turbulent 3D Features. IEEE Trans. Visualization and Computer Graphics Apr.-June;1997 3(2):129–141.

22. Silver D, Wang X. Tracking Scalar Features in Unstructured Datasets. Proc. IEEE Visualization Conf 1998:79–86.

23. Sircar JK, Cerbrian JA. Application of Image Processing Techniques to the Automated Labelling of Raster Digitized Contours. Proc. Int'l Symp. Spatial Data Handling 1986:171–184.

24. Sutton PM, Hansen CD. Isosurface Extraction in Time-Varying Fields Using a Temporal Branch-on-Need Tree (T-BON). Proc. IEEE Visualization Conf 1999:147–154.

25. Takahashi S, Ikeda T, Shinagawa Y, Kunii TL, Ueda M. Algorithms for Extracting Correct Critical Points and Constructing Topological Graphs from Discrete Geographical Elevation Data. Computer Graphics Forum 1995;14(3):181–192.

26. Tarasov SP, Vyalyi MN. Construction of Contour Trees in 3D in $O(N \log N)$ Steps. Proc. ACM Symp. Computational Geometry 1998:68–75.

27. van Kreveld MJ, van Oostrum R, Bajaj CL, Pascucci V, Schikore D. Contour Trees and Small Seed Sets for Isosurface Traversal. Proc. ACM Symp. Computational Geometry 1997:212–220.

28. Weigle C, Banks DC. Extracting Iso-Valued Features in 4-Dimensional Scalar Fields. Proc. IEEE Symp. Volume Visualization 1998:103–110.

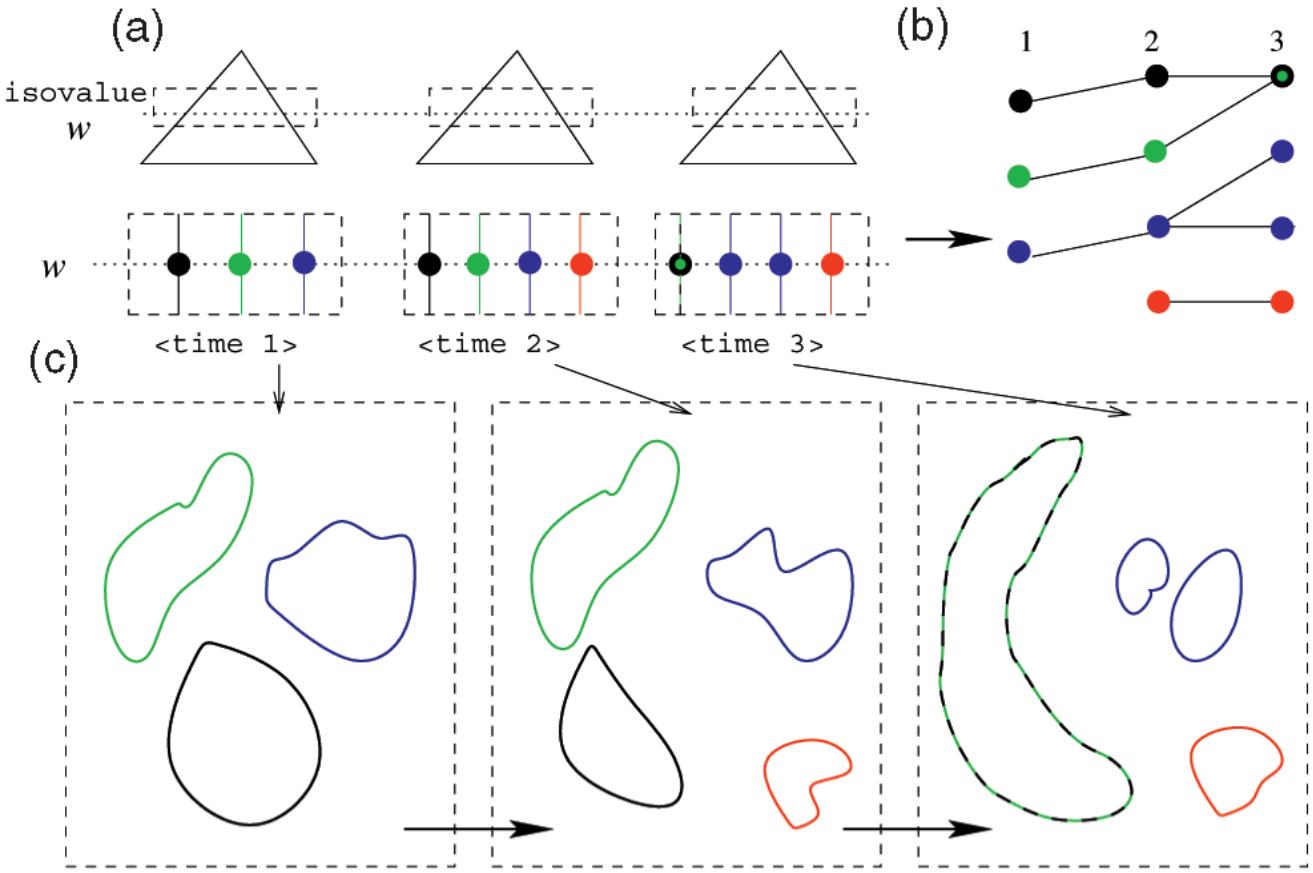


Fig. 1.

Evolution of three isosurfaces. An intersection point on an edge of a contour tree represents a contour component. (a) By using precomputed correspondence information in time-dependent contour trees and (b) a graph representing the topology changes of time-varying isosurfaces is immediately constructed for any selected isovalue. (c) Seed sets generated from the contour trees are used for rapid extraction of the time-dependent surfaces of segmented contours. Note that colors represent correspondence among intersection points in contour trees, nodes of the graph, and contour components.

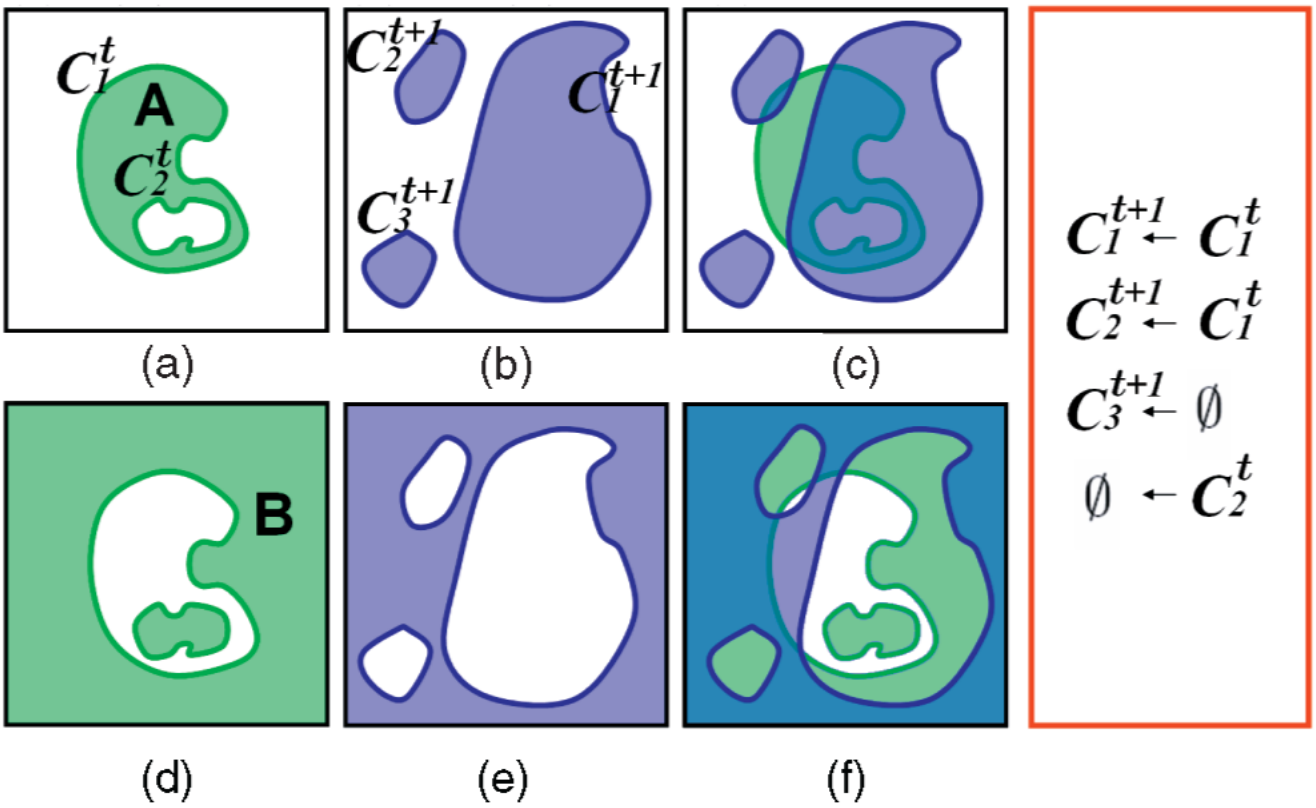


Fig. 2.
Contour correspondence test of two successive isosurfaces. (a) and (b) Upper objects. (d) and (e) Lower objects. (c) and (f) Check overlaps of upper and lower objects, respectively.

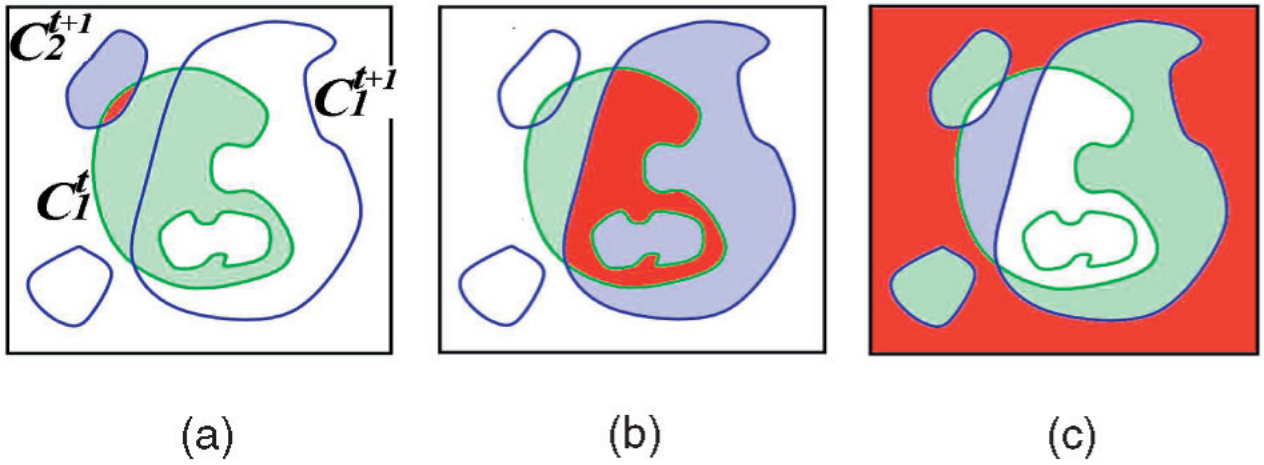


Fig. 3.

Test for the degree of overlap. The overlapping region is colored red. The remainder of objects are colored green and blue in time t and $t + 1$, respectively. (a) Overlap between upper objects of C_1^t and C_2^{t+1}. (b) Overlap between upper objects of C_1^t and C_1^{t+1}. (c) Overlap between lower objects of C_1^t and C_1^{t+1}. Note that the degree of overlap in (a) is small, while the degree in (b) and (c) is reasonably large. If we require significant overlap (e.g., $\epsilon = 0.3$), C_1^t does not correspond to C_2^{t+1}, but C_1^t corresponds to C_1^{t+1}.

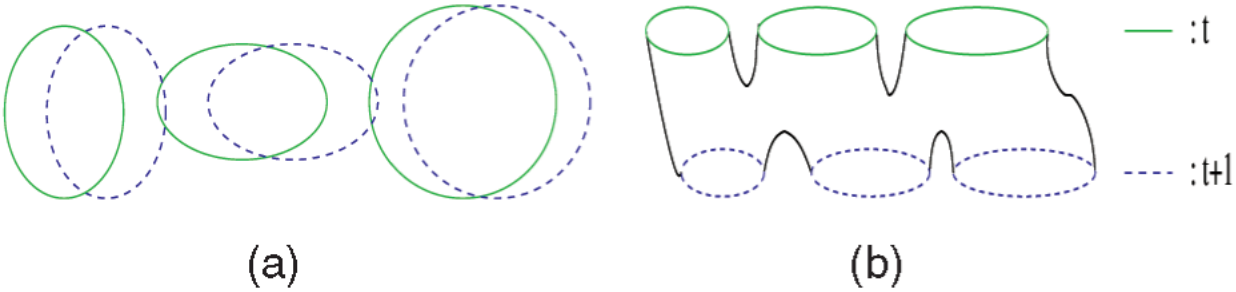


Fig. 4.

An example which shows the correspondence definition based on the degree of overlap gives better matching than the definition based on the contour continuity. (a) Three contours in time *t* and *t* + 1. Each contour in time *t* is slightly translated to the right in time *t* + 1. (b) A contour defined from higher dimensional function interpolated over time [14]. Each contour in time *t* corresponds to every contour in time *t* + 1 even though their interdistance is big.

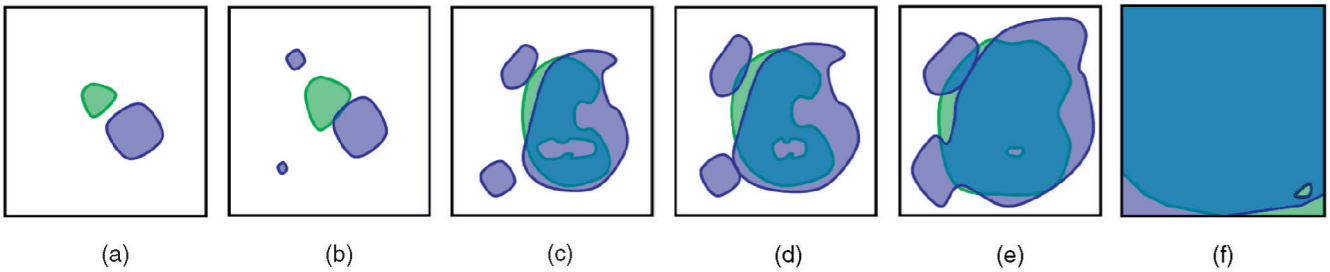


Fig. 5.
The growth of upper objects in time t (green) and $t + 1$ (blue) as an isovalue decreases. Collision occurs between green and blue objects at (b) and at (c). Between (a) and (b), two blue objects are created. Between (b) and (c), a contour on a green object is split. Blue objects merge at (d) and (e).

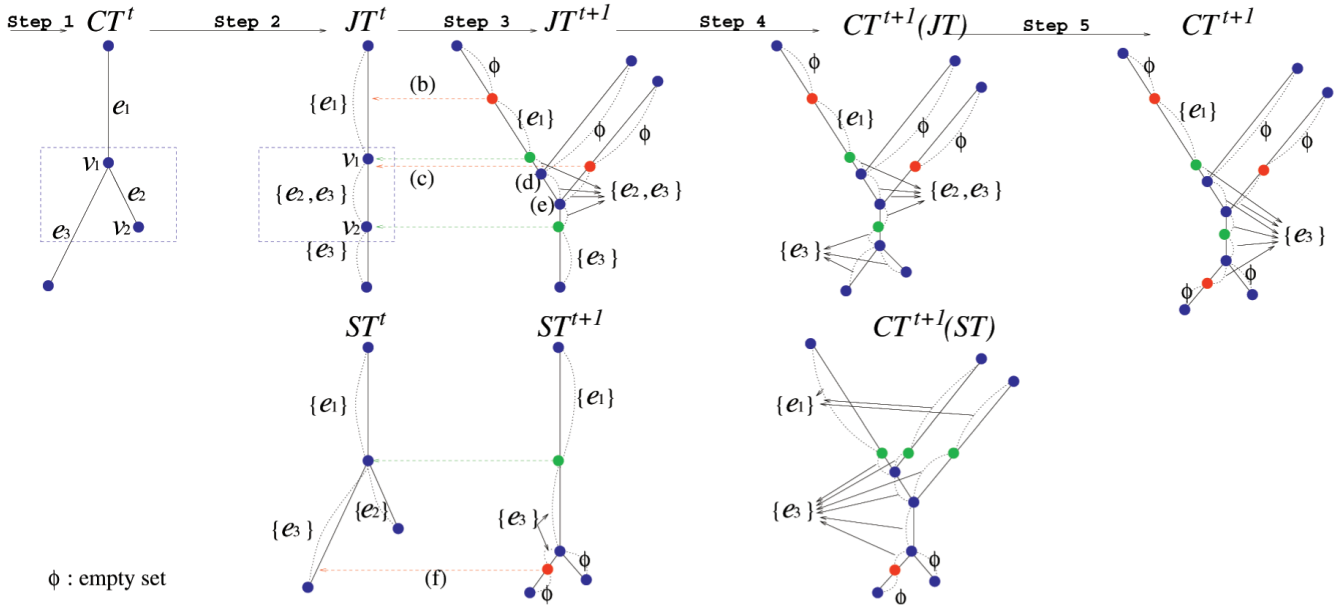


Fig. 6.
Five steps for computing correspondence of contours in two contour trees.

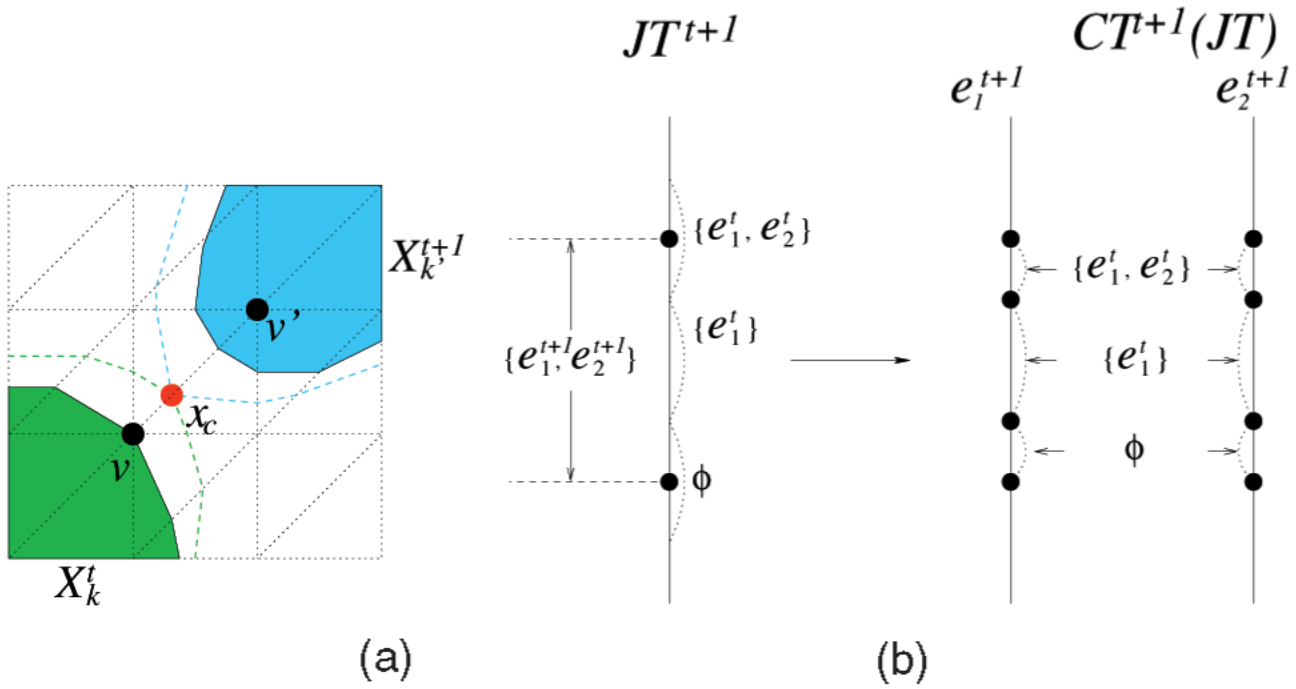


Fig. 7.

(a) Collision of two objects from time t and $t + 1$ should occur at a point x_c on an edge of a mesh with a PL function. x_c is computed from the positions of v and v' and the values $f^t(v)$, $f^{t+1}(v)$, $f^t(v')$, and $f^{t+1}(v')$. (b) Example of Step 4: The labels $\{e_1^t, e_2^t\}$, $\{e_1^t\}$, and $\varnothing$ of an edge in JT^{t+1} are copied to the corresponding edges, e_1^{t+1} and e_2^{t+1}, in CT^{t+1}.

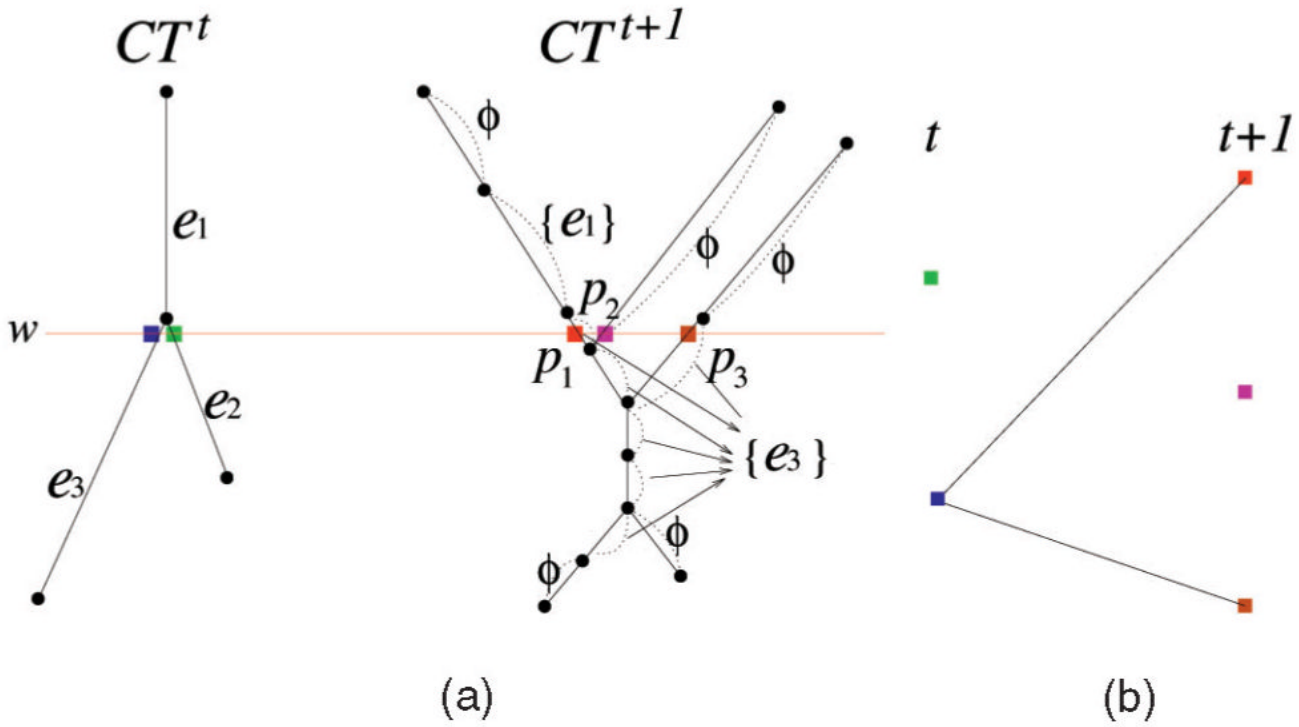


Fig. 8.
(a) Checking correspondence by using labeled contour trees. (b) Topology Change Graph. Note that colors are used to show the correspondence between the intersection points in contour trees and the nodes in TCG. This result is consistent with the correspondence relationship in Fig. 2.

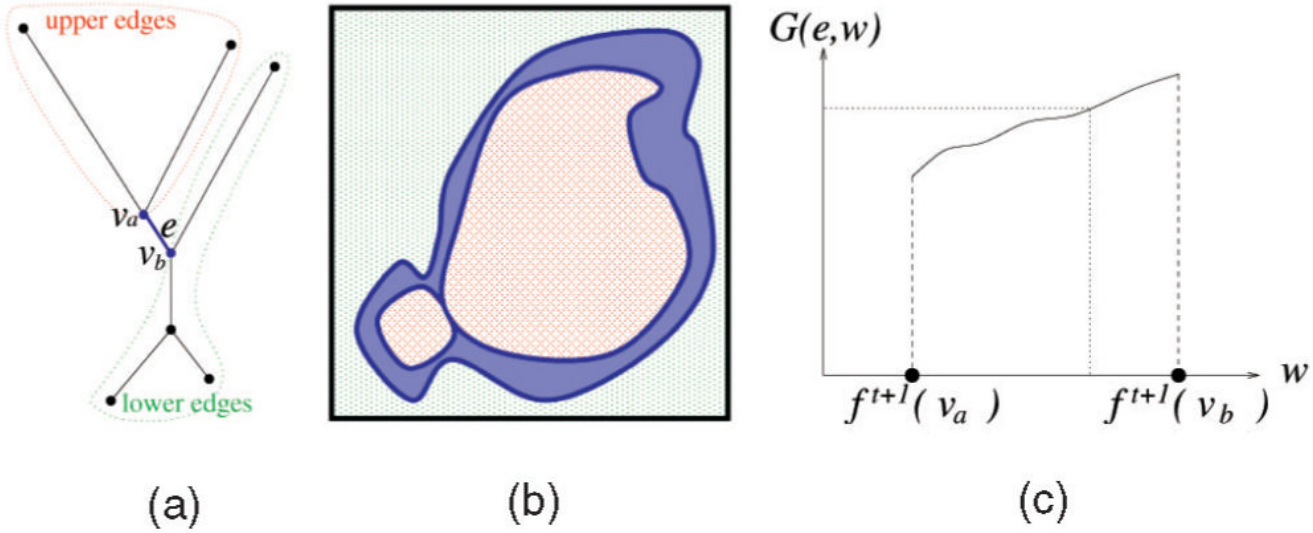


Fig. 9.

Quantification for contours in CT^{t+1} of Fig. 6. (a) an edge e divides a contour tree into a set of upper edges and a set of lower edges. (b) The regions covered by a class of continuous contours corresponding to e (blue), upper edges (red), and lower edges (green). (c) The function $G(e, w)$ represents a geometric quantity of a contour on e for an isovalue w.

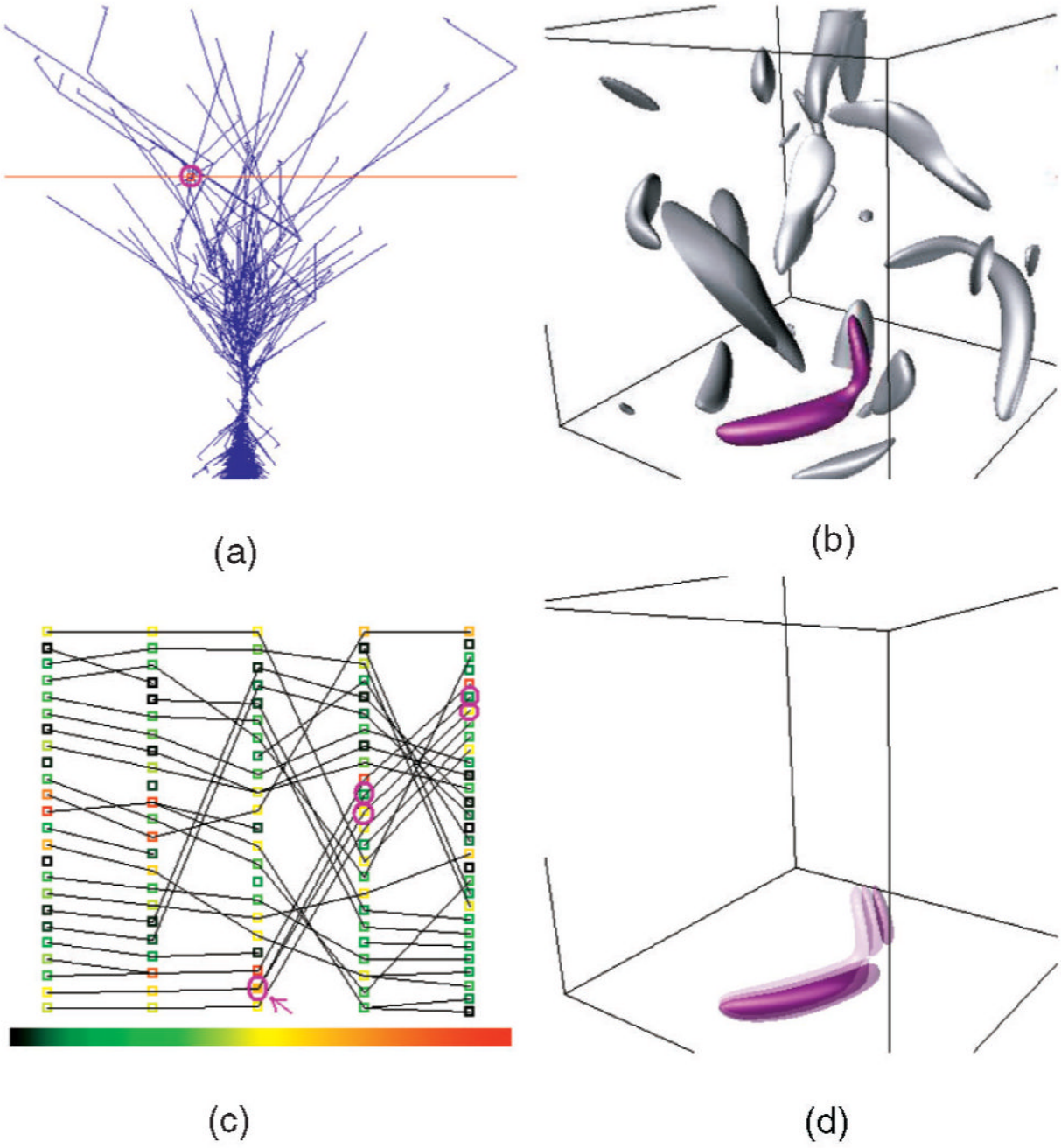


Fig. 10.
Simulations of turbulent vortex structures. The color of a node in (c) represents the surface area of a contour which corresponds to the node. The marked nodes correspond to segmented contours. (d) Transparent display of contour evolution. (a) Contour tree. (b) Contour segmentation. (c) Topology change graph. (d) Segmented contour tracking.

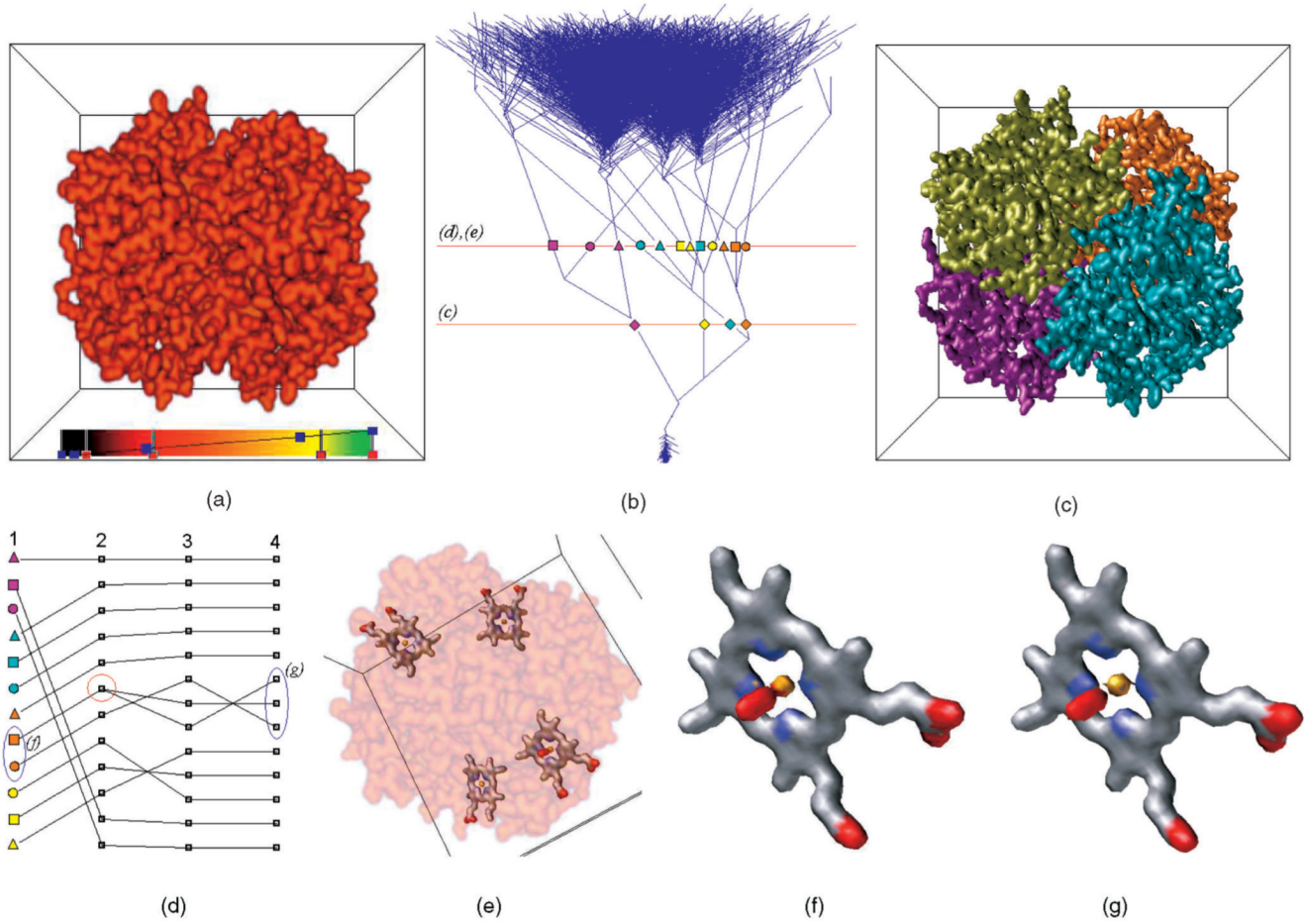


Fig. 11.

Visualization of a hemoglobin molecule and its dynamics. In (c), each chain is segmented and colored. (f) and (g) show a heme group composed of carbon (gray), nitrogen (blue), iron (yellow), hydrogen (not shown), and oxygen (red) atoms with different timesteps. Note that oxygen bound to iron in (f) is released in (g). This phenomena is detected in the red circle of (d). (a) Volume rendering. (b) Contour tree. (c) Segmentation of four polypeptide chains. (d) Topology change graph. (e) Four heme groups. (f) The heme group at time 1. (g) The heme group at time 4.