# Time-Varying Contour Topology

Bong-Soo Sohn
Department of Computer Sciences
University of Texas at Austin
bongbong@cs.utexas.edu

Chandrajit Bajaj
Department of Computer Sciences
University of Texas at Austin
bajaj@cs.utexas.edu

## ABSTRACT

The contour tree has been used to compute the topology of isocontours, generate a minimal seed set for accelerated isocontour extraction, and provide a user interface to segment individual contour components in a scalar field. In this paper, we extend all 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 isocontours is constructed in real-time for any selected isovalue using the pre-computed correspondence information. Quantitative properties such as surface area and volume changes of contour components are computed and labelled on the graph. This topology change graph helps users to easily detect the significant topological and geometric changes in time-varying isocontours. The graph is also used as a user interface to quickly segment, track and visualize the evolution of any selected contour component over time.

## Keywords

Contour Tree, Level Set Topology, Feature Tracking, Time-Varying Volume Visualization

## 1. INTRODUCTION

Scientific simulations of today are increasingly generating densely sampled time-varying fields. 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 is then visualized. Isocontouring or volume rendering is a common way to visualize the evolution of features in data. However, just rendering a sequence of volumes or isocontours does not explicitly provide the dynamic features. In this paper, we describe an algorithm to compute the correspondence information of contours for all isovalues in time-varying scalar fields. This allows to interactively track the topology changes of time-varying isocontours and is used to extract dynamic features in the data set. For instance, a cosmological simulation generates time-varying density and temperature fields of the universe to understand the formation of galaxy clusters. The images and movies of the data are available in our website [10]. We are able to detect when and where an individual galaxy cluster is created, vanished, split and merged with other clusters, and to measure how much a galaxy cluster grows or shrinks. Other examples of feature tracking can be found in [14, 20].

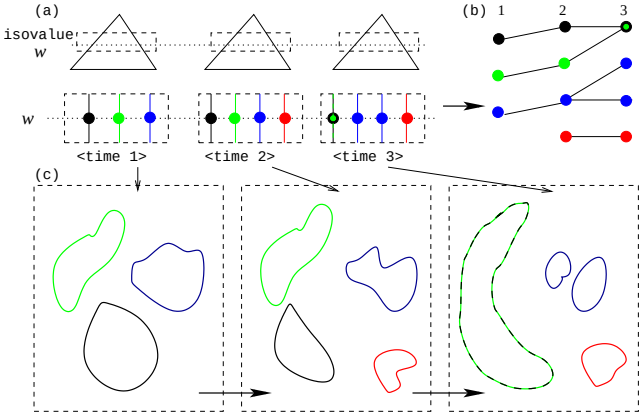



**Figure 1: Evolution of three consecutive isocontours. An intersection point on an edge of a contour tree corresponds to a contour component. By using pre-computed correspondence information in time dependent contour trees (a), a graph representing the topology changes of time-varying isocontours is constructed in real-time for any selected isovalue (b). Seed sets generated from the contour trees are used to quickly extract the time dependent surfaces of segmented contours (c). Note that colors are used for showing the correspondence.**

The *Contour Tree* (CT) [6, 16, 25] is useful for the visualization of a scalar field. First, CT provides topological structures of a scalar field, which are not easily obtained from rendering techniques. Second, CT is used to generate a minimal seed set for efficient isocontouring. Third, CT provides a user interface to segment and render an individual connected component of an isocontour. Each edge of CT represents a single connected component of an isocontour. Clicking a point on an edge yields fast extraction of the corresponding isocontour component by propagation from a seed cell computed from the contour tree. However, CT is used for a single scalar field and the benefits are limited to the visualiztion of a static scalar field. Our main objective is to extend all the benefits of CT to time-varying data visualization.

The input is time-varying scalar fields $\{f^1, ..., f^T\}$ defined on a simplicial mesh $M$. Time dependent contour trees $\{CT^1, ..., CT^T\}$ can be computed for each timestep. Each contour tree is

laid 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, let the intersection points between a contour tree $CT^t$ and a line $y = w$ be $P^t = \{p_1^t, ..., p_{n_t}^t\}$. Each point $p_j^t$ represents a connected component $C_j^t$ of an isocontour $I^t$, where $I^t = \{C_1^t, ..., C_{n_t}^t\}$. We call a connected component of an isocontour, $C_j^t$, a *contour*. Now, we consider two consecutive contour trees, $CT^t$ and $CT^{t+1}$, and an arbitrary isovalue $w$. $P^t$ and $P^{t+1}$ can be obtained from the contour trees. We represent correspondence information $p_j^{t+1} \leftarrow \{p_{\pi(1)}^t, ..., p_{\pi(n_j)}^t\}$ to mean that each of $C_{\pi(1)}^t, ..., C_{\pi(n_j)}^t$ corresponds (evolves) to $C_j^{t+1}$.

Since $w$ is an arbitrary floating point value in an interactive application, it is difficult to compute the correspondence information for every possible value of $w$ as a preprocessing. Fortunately, the correspondence information of contours has coherence such that we can store the same information over a range of isovalues. Let each edge of $CT^t$ be labelled as $e_j^t$. The goal is to assign a set of edges for time $t$, $E^t = \{e_{\pi(1)}^t, ..., e_{\pi(n)}^t\}$, to a corresponding edge $e_{j'}^{t+1}$ with the value range $[f_a, f_b]$ in $CT^{t+1}$, which is represented as $(e_{j'}^{t+1}, [f_a, f_b]) \leftarrow \{e_{\pi(1)}^t, ..., e_{\pi(n)}^t\}$. This means that for any isovalue $w \in [f_a, f_b]$, an intersection point $p_{j'}^{t+1}$ on $e_{j'}^{t+1}$ has correspondence with intersection points $p_{\pi(1)}^t$,..., $p_{\pi(n)}^t$ on $e_{\pi(1)}^t, ..., e_{\pi(n)}^t$, and hence $C_{j'}^{t+1} \leftarrow \{C_{\pi(1)}^t$,...,$C_{\pi(n)}^t\}$.

Once this correspondence information is computed as a preprocessing, a graph representing the topology changes of time-varying isocontours can be constructed in real-time for any selected isovalue. This graph is called a *Topology Change Graph (TCG)*. Let $P^{t+1} = \{p_1^{t+1}, ..., p_{n_{t+1}}^{t+1}\}$ and $p_c^{t+1} \leftarrow \{p_{(1,c)}^t, ..., p_{(n_c,c)}^t\}$ for an isovalue $w_0$, where $c = 1, ..., n_{t+1}$. TCG is constructed by creating nodes for every intersection point and connecting each pair of $(p_{(j,c)}^t, p_c^{t+1})$ for the time sequence $t = 1$ , ... , $T - 1$ and $j = 1$ , ... , $n_c$, as shown in Figure 1. An additional quantitative information such as surface area and volume of each contour can also be computed and labelled on TCG. Some of the applications of TCG are as follows :

- **Dynamic Structure Extraction** : One can easily determine the topology changes of time-varying isocontours ( eg. *merge, split, create, vanish,* and *genus change* of contours).
- **Feature Tracking** : One can quickly segment, track, quantify and visualize the evolution of any individual contours.

The main contributions of this paper are to (i) define the temporal correspondence of contours, describe an algorithm to compute the correspondence information in time-varying contour trees and analyze its time complexity, (ii) apply the correspondence information to construct TCG in real-time for any selected isovalue, and (iii) present real-life applications of TCG for dynamic structure extraction and feature tracking.

The remainder of this paper is organized as follows. After reviewing related works in section 2, we define the temporal correspondence of contours and test conditions in section 3. In section 4 and 5, we describe an algorithm to construct contour trees and compute correspondence information in time dependent contour trees. The time complexity is analyzed. In section 6, we describe how to construct a topolgy change graph. In section 7, we compute quantitative properties of contours in a contour tree. In section 8, an interactive system to segment and track the evolution of interesting contours is built. Experimental results are presented in section 9. Finally, in section 10, we conclude this paper.

## 2. RELATED WORK

In this section, we review previous works on contour trees, time-varying data visualization, and feature tracking.

**Contour Tree** CT has been used in various fields such as image processing and GIS [21, 23]. Our main interest is to use it in visualization. van Kreveld et al. [25] described an $O(mlogm)$ and $O(m^2)$ algorithm to construct a contour tree from a 2D and 3D scalar field defined on a simplicial mesh with $m$ elements and $n$ vertices. The computed contour tree is used to generate a minimal seed set for optimal isocontour extraction [2]. Tarasov and Vyalyi [24] improved the time complexity to $O(m\log m)$ in the 3D case. Carr, Snoeyink and Axen [6] simplified the 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 [15] computed betti numbers of contours to distinguish different topology of contours within an edge of CT. The divide and conquer approach [16] allows output-sensitive construction [8] of contour trees and easy extension to parallel implementation. Carr and Snoeyink [5] used CT as an interface to display topological structures of isocontours and segment individual contours in a scalar field. They computed path seeds for each edge, which generate a seed cell necessary for rapidly extracting a selected contour in run-time. CT is also used for preserving the topology of isosurfaces during progressive simplification of tetrahedral meshes where a function is defined [9].

**Isocontouring 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 [18] [22] are used for minimizing unnecessary I/O access and supporting out-of-core isosurface extraction [7] in time-varying fields. The high dimensional isocontouring approach [3, 26] considers the time dependent data in 4-dimensional space $f(x, y, z, t)$. They first extract a 3-dimensional solid mesh $f(x, y, z, t) = w$. Then, an isocontour at time $t_0$, $t(x, y, z) = t_0$, 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 [1] computes and shows geometric and topological properties such as surface area, volume, and gradient integral of isocontours over all isovalues and timesteps. A similar interface called *contour plane* [13] displays the number of contours over all isovalues and timesteps in a 2D plane. This represents the amount of topological changes in time-varying isocontours.

**Feature Tracking** Silver et al. [19, 20] defined a feature in a volume as a region of interests which satisfies a predefined thresholding criteria. After feature extraction, they perform a correspondence matching test of features based on the degree of overlap to track and quantify the movement of each isolated feature over time. Dynamic events of the features are classified as continuation, creation, dissipation, bifurcation and amalgamation. High dimensional isocontouring can be applied to feature tracking [12].

## 3. CONTOUR CORRESPONDENCE

Given two consecutive isocontours, $I^t = \{C_1^t, ..., C_{n_t}^t\}$ and $I^{t+1} = \{C_1^{t+1}, ..., C_{n_{t+1}}^{t+1}\}$, a correspondence test determines whether a contour $C_{k_t}^t$ corresponds (evolves) to a contour $C_{k_{t+1}}^{t+1}$. There is no absolute rule for the test because it is impossible to know how the isocontour changes between the two timesteps unless a specific assumption is made. We provide our own rule for

the correspondence test and then justify its validity.

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 $X^t(w) = \{X_1^t, \ldots, X_n^t\}$. $B(X_k^t(w))$, the border of an object $X_k^t(w)$, is defined as an intersection of the object and the isocontour, $X_k^t(w) \cap I(w)$. The border of each object has one or more than one contour, $B(X_k^t(w)) = \{C^t_{(X_k^t,1)}, \ldots, C^t_{(X_k^t,l)}\}$. Similarly, we can define $Y^t(w) = \{x |\ f^t(x) \leq w\}$, $Y_k^t$, and $C^t_{(Y_{k'}^t,l')}$. We term $X_k^t$ an upper object and $Y_{k'}^t$ a lower object for convenience.

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 $Y_{a'}^t$, and $C_{k'}^{t+1}$ is on the border of $X_b^{t+1}$ and $Y_{b'}^{t+1}$. If two upper objects $X_a^t$ and $X_b^{t+1}$ overlap each other and so do lower objects $Y_{a'}^t$ and $Y_{b'}^{t+1}$, then we call $C_k^t$ corresponds to $C_{k'}^{t+1}$, which is denoted as $C_{k'}^{t+1} \leftarrow C_k^t$. This definition is formally stated in :

**Definition 1 : Contour Correspondence**

Suppose $C_k^t \in B(X_a^t)$ , $C_k^t \in B(Y_{a'}^t)$ ,$C_{k'}^{t+1} \in B(X_b^{t+1})$ , $C_{k'}^{t+1} \in B(Y_{b'}^{t+1})$.

$$(X_a^t \cap X_b^{t+1}) \neq \emptyset \text{ and } (Y_{a'}^t \cap Y_{b'}^{t+1}) \neq \emptyset \Longleftrightarrow C_{k'}^{t+1} \leftarrow C_k^t.$$

Figure 2 shows an example of the correspondence test. There are two contours $C_1^t$ and $C_2^t$ at time $t$, and three contours $C_1^{t+1}$ , $C_2^{t+1}$ , $C_3^{t+1}$ at time $t+1$. As can be seen in Figure 2, upper objects of $C_1^{t+1}$ and $C_2^{t+1}$ overlap with 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 with the upper object of $C_1^t$ and $C_2^t$. The lower object of $C_2^t$ does not overlap with other lower objects. Therefore, $C_1^{t+1} \leftarrow C_1^t$ , $C_2^{t+1} \leftarrow C_1^t$ , and $C_3^{t+1} \leftarrow \emptyset$. $\emptyset \leftarrow C_2^t$.

To justify the validity of the condition for the correspondence test, we make an assumption on the movement of contours over time. Let $w$ be an isovalue. We first define $sign_t(x)$, a sign of a point $x$ on the domain in the function $f^t$ (Figure 3 (a)).

$$sign_t(x) = 1 \text{ , if } f^t(x) > w$$
$$sign_t(x) = -1 \text{ , if } f^t(x) < w$$
$$sign_t(x) = 0 \text{ , if } f^t(x) = w$$

Consider the case when $f^t$ and $f^{t+1}$ are placed on the same domain. We define $signchange_{(t,t+1)}(x) = sign_t(x) \cdot sign_{t+1}(x)$, indicating whether the sign of $x$ changes over time. The whole sign change area can be decomposed into a set of sign change subregions $r_1, ..., r_n$ where each $r_k$ is enclosed by contour segments. Intuitively, each sign change subregion $r_k$ is considered as a homogenious region whose *signchange* value is negative (inside the region) or zero (the border of the region) as shown in Figure 3 (b).

Based on sign change subregions, we are able to estimate the movement of contours (Figure 3(c)). An isocontour at $t+1$ may partition a contour at $t$ into several segments. We define $S_i(C_k^t)$ as an $i$-th segment of $C_k^t$, which is separated by $I^{t+1}$. In a similar way, $S'_{i'}(C_{k'}^{t+1})$ is defined as an $i'$-th segment of $C_{k'}^{t+1}$ separated by $I^t$. We assume that $S_i(C_k^t)$ moves to $S'_{i'}(C_{k'}^{t+1})$ if $S_i(C_k^t)$ and $S'_{k'}(C^{t+1})$ are on the border of the same sign change subregion, and vice versa. If the border contains only $C_{k'}^{t+1}$, $C_{k'}^{t+1}$ is created as a point between $t$ and $t+1$, and the point grows into $C_{k'}^{t+1}$. If the border contains only $C_k^t$, $C_k^t$ shrinks into a point between $t$ and $t+1$, and disappears. The contour segments and their movement are shown in Figure 4 (a) and (b). This assumption is reasonable because the sign change subregion is the only possible area which the contour segment on the border of the subregion can move through, if $f^{t_0}$ is linearly interpolated between $f^t$ and $f^{t+1}$, where

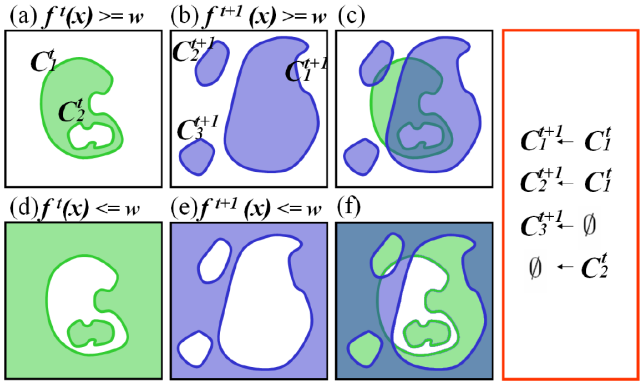



**Figure 2: Contour correspondence test of two consecutive isocontours. (a)(b) Upper objects. (d)(e) lower objects. (c) and (f) check overlaps of upper and lower objects respectively.**

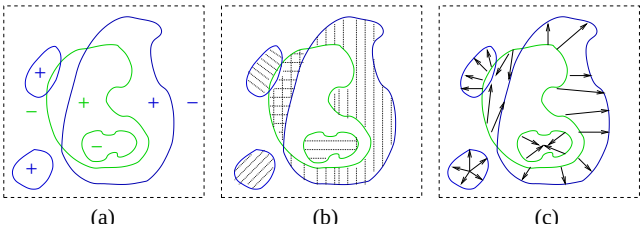


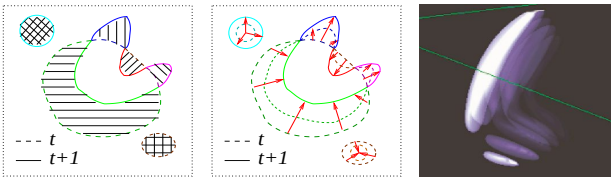

**Figure 3: (a) Consecutive isocontours at time $t$ (green) and $t+1$ (blue). Each bounded region is marked as either '+' or '-' based on its *sign*. (b) Each sign change subregion is marked with lines. (c) Contour movement. Points on a contour $C_k^t$ at time $t$ move toward corresponding points on the contours at time $t+1$ which share the same sign change subregion with $C_k^t$.**

$t < t_0 < t+1$.

Our rule for the correspondence test guarantees to decide (i) $C_{k'}^{t+1} \leftarrow C_k^t$ if $S_i(C_k^t)$ moves to a segment $S'_{i'}(C_{k'}^{t+1})$, (ii) $C_{k'}^{t+1} \leftarrow \emptyset$ if $C_{k'}^{t+1}$ is newly created, (iii) $\emptyset \leftarrow C_k^t$ if $C_k^t$ disappears, based on the movement assumption we made in the previous paragraph. The properties (i),(ii) and (iii) shows the validity of the definition 1. A brief proof is as follows. Assume $S_i(C_k^t)$ moves to $S'_{i'}(C_{k'}^{t+1})$. That means a sign change subregion $r$ has $S_i(C_k^t)$ and $S'_{i'}(C_{k'}^{t+1})$ on its border. Let's assume a sign changes from negative to positive in $r$. The case of a sign change from positive to negative is symmetric. Then $X_a^t$ containing $C_k^t$ on its border is laid just outside of the sign change subregion $r$, and $X_b^{t+1}$ contains $r$. Therefore, there is an overlap between $X_a^t$ and $X_b^{t+1}$ at least on $S_i(C_k^t)$. The case of lower objects is also same. $Y_{b'}^{t+1}$ is laid just outside of $r$ and $Y_{a'}^t$ contains $r$. Therefore, there is an overlap between $Y_{a'}^t$ and $Y_{b'}^{t+1}$ at least on $S'_{i'}(C_{k'}^{t+1})$. This means $C_{k'}^{t+1} \leftarrow C_k^t$. In the case of (ii) and (iii), the associated sign change subregion $r$ is covered by only one upper object or lower object. Overlapping between either upper objects or lower objects does not happen. Therefore, $C_{k'}^{t+1} \leftarrow \emptyset$ (ii), or $\emptyset \leftarrow C_k^t$ (iii). The converses of (i),(ii) and (iii) are true for most cases, but some weird cases do not satisfy the converses of (i),(ii) and (iii).

## 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 dissipated.




(a) (b) (c)

**Figure 4: (a) Each contour segments are colored differently. (b) The movement of each segment through sign change subregion. Dashed contours are defined from interpolated function at $t_0 \in (t, t+1)$. A contour at $t$ continuously evolves to a contour at $t+1$. (c) Contour evolution.**

Define a contour class as a maximal set of continuous contours which do not contain critical points. $E$ consists of edges connecting two critical points where a contour class is created and destroyed.

Our algorithm for computing correspondence information between consecutive 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) to build CT. Since construction of JT and ST is symmetric, we only describe the algorithm for JT construction.

Start from the maximum function value $w = max(f)$. We continuously decrease an isovalue $w$ and mark regions $X(w) = \{x | f(x) \geq w\} = \{X_1, X_2, ...X_n\}$ on the domain space, where $X_k$ is a connected component. Each connected component of the marked regions is conceptually same as an upper object. As an isovalue passes through the function value of a local maximum, called *upper leaf*, a new component $X_{n+1}$ is created. At this moment, a JT node for the upper leaf is created. In the case of ST construction, it is called *lower leaf*. As an isovalue passes through a joining saddle point, called $join$, two or more components are merged into one. *split* is defined in a similar way as $join$. 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 each joining component created. When $w$ reaches the global minimum function value, a JT node is created and connected to the latest joining node.

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

# 5. CORRESPONDENCE COMPUTATION

## 5.1 Algorithm Overview

In this section, we aim to design an algorithm to compute the contour correspondence over all isovalues as a preprocessing. The problem is formally stated as follows. Each edge of $CT^t$ is labeled as $e_j^t$. The goal is to assign a set of edges $E_k^t = \{e_{\pi(1)}^t, \ldots, e_{\pi(n)}^t\}$ to a corresponding edge $e_{k'}^{t+1}$ with a function range $[f_a, f_b]$ in $CT^{t+1}$, which is represented as $(e_{k'}^{t+1}, [f_a, f_b]) \leftarrow \{e_{\pi(1)}^t, \ldots, e_{\pi(n)}^t\}$. This means any intersection point on $e_{k'}^{t+1}$ with an isovalue $w_0 \in [r_a, r_b]$ corresponds to intersection points on $e_{\pi(1)}, \ldots, e_{\pi(n)}$ with $w_0$. The correspondence information needs to be computed for every edge in $CT^{t+1}$.

As can be noted, definition 1 and the process to construct join/split trees have an interesting relationship. The process of JT construction is very similar to checking an overlap of two upper objects, $(X_a^t(w) \cap X_b^{t+1}(w)) \neq \emptyset$, as the threshold value $w$ decreases from the highest function value. Given $f^t$ and $f^{t+1}$ on the same domain, we start from the highest value of $f^t$ and $f^{t+1}$, 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 *objects* $X^t$ and $X^{t+1}$, which are created and merged each other. If an upper object $X_a^t$ and $X_b^{t+1}$ collides at a point $x_c$ and isovalue $w_0$, $X_a^t$ and $X_b^{t+1}$ starts to have an overlap area at isovalue $w_0$. Objects $X_a^t$ and $X_b^{t+1}$ always grows as the isovalue decreases. Therefore two objects always have an overlap area after collision between the two objects. This can be formulated as $X_a^t(w) \cap X_b^{t+1}(w) \neq \emptyset$, where $w \leq w_0$. This relationship is exactly same for the split tree construction and checking $(Y_{a'}^t(w) \cap Y_{b'}^{t+1}(w)) \neq \emptyset$. Using this property, we use JTs and STs of $f^t$ and $f^{t+1}$ to compute the overlap information of upper and lower objects over all isovalues. The overlap information is used to construct $CT^{t+1}$ where the temporal correspondence information of contours are computed.

## 5.2 Algorithm Details

Each edge of $CT^t$ has a unique id $e^t$. Any point $p^t$ on an edge $e^t$ corresponds to a contour $C^t$ and vice versa. This is represented as $E(C^t) = e^t$. We define various forms of join and split trees and contour trees. Each edge $e = (v_a, v_b)$ of the original $JT^t/ST^t/CT^t$ and $JT^{t+1}/ST^{t+1}/CT^{t+1}$ is decomposed into subedges $(v_a, v_1)$,$(v_1, v_2)$ ,..., $(v_k, v_b)$ such that every point on a subedge has correspondence with the same set, termed ESET, of edges in $CT^t$. During the edge decomposition, nodes $v_1, ..., v_k$ are created. It is possible that the edge decomposition is not necessary. Depending on the meaning of the 'correspondence' and ESET, different forms of JT/ST/CT, namely $JT_E^t$, $JT_C^{t+1}$, $CT_{C_{JT}}^{t+1}$, and $CT_C^{t+1}$ are defined as follows.

- $JT_E^t$ : Any point $p$ on a subedge $s$ of $JT_E^t$ corresponds to an upper object $X_p^t$. The border of $X_p^t$ may contain several contours, $B(X_p^t) = \{C_1^t, ..., C_k^t\}$. For any point on the subedge $s$, $ESET(s) = \{E(C_1^t), ..., E(C_k^t)\}$.
- $JT_C^{t+1}$ : Any point on a subedge $s$ of $JT_C^{t+1}$ corresponds to an upper object $X_i^{t+1}$. There are a set of objects $X_1^t, ..., X_k^t$ which overlap with $X_i^{t+1}$. Let $B(X_1^t) \cup ... \cup B(X_k^t) = \{C_1^t, ..., C_{k'}^t\}$. $ESET(s) = \{E(C_1^t), ..., E(C_{k'}^t)\}$.
- $CT_{C_{JT}}^{t+1}$ : Any point on a subedge $s$ of $CT_{C_{JT}}^{t+1}$ corresponds to a contour $C_i^{t+1}$. There exists an upper object $X_{i'}^{t+1}$ containing $C_i^{t+1}$ on its border. There are a set of objects $X_1^t, ..., X_k^t$ which overlap with $X_{i'}^{t+1}$. Let $B(X_1^t) \cup ... \cup B(X_k^t) = \{C_1^t, ..., C_{k'}^t\}$. $ESET(s) = \{E(C_1^t), ..., E(C_{k'}^t)\}$.
- $CT_c^{t+1}$ : Any point on a subedge $s$ corresponds to a contour $C_i^{t+1}$. There exist an upper object $X_{a'}^{t+1}$ and a lower object $Y_{b'}^{t+1}$ containing $C_i^{t+1}$ on their borders. There are a set of contours $\{C_1^t, ..., C_k^t\}$ such that $X(C_j^t) \cap X(C_i^{t+1}) \neq \emptyset$ and $Y(C_j^t) \cap Y(C_i^{t+1}) \neq \emptyset$, $j = 0, 1, ..., k$. $X(C_j^t)$ and $Y(C_j^t)$ returns $X_a^t$ and $Y_b^t$ respectively, where $C_j^t \in B(X_a^t)$ and $C_j^t \in B(Y_b^t)$. $ESET(s) = \{E(C_1^t), ..., E(C_k^t)\}$.

The goal of this section is to construct $CT_c^{t+1}$. The algorithm consists of five steps : the construction of (1) $CT^t$,$JT^t/ST^t$ , $CT^{t+1}$,$JT^{t+1}/ST^{t+1}$ , (2) $JT_E^t/ST_E^t$ , (3) $JT_C^{t+1}/ST_C^{t+1}$ , (4) $CT_{C_{JT}}^{t+1}/CT_{C_{ST}}^{t+1}$ , and (5) $CT_C^{t+1}$. Each step except the first one uses the information computed from the previous step. '/' represents symmetric relationship. Therefore, we describe algorithms only for the cases of $JT_E^t$, $JT_C^{t+1}$, $CT_{C_{JT}}^{t+1}$ in step 2, 3 and 4, respectively. Figure 5 shows the growth of upper objects in $f^t$ and

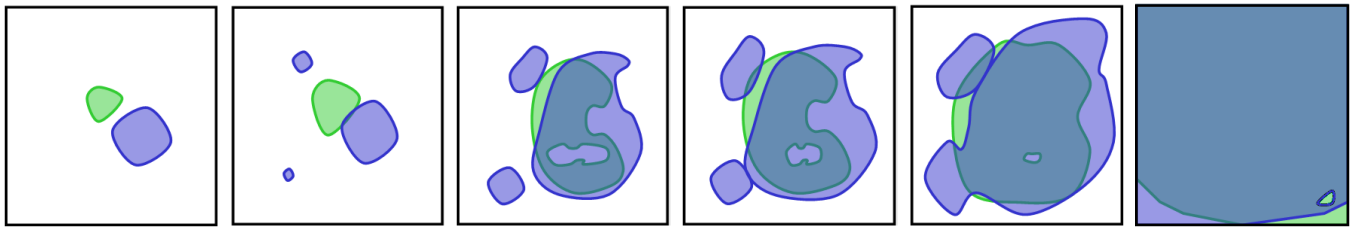



**Figure 5: The growth of upper objects in time $t$ (green) and $t+1$ (blue) as an isovalue decreases.**

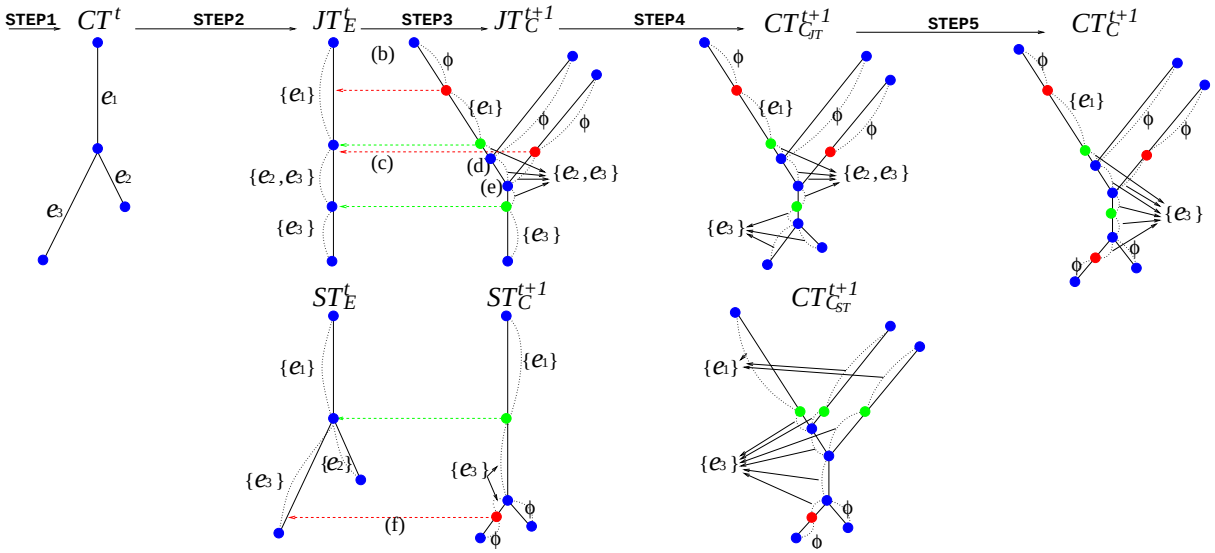



**Figure 6: Five steps for computing correspondence of contours in consecutive contour trees.**

$f^{t+1}$ on the same domain when $w$ is continuously decreased from the maximum value. We use the same function $f^t$ and $f^{t+1}$ for Figure 2, 3, 5, 6 and 9. Figure 6 shows how the trees are constructed for each five step. The following paragraphs describe the algorithm of each step in detail.

We use the algorithms described in [6] for step 1. The second step is to construct $JT_E^t$. The vertices of the domain mesh are sorted in an increasing order and stored in the array $va$. The functions 'CreateNode' and 'Connect' generate the nodes and edges of the output tree. The node needs to be stored with its function value. The ESET of an edge is determined when the node which has the higher function value is created. E$(v_1, v_2)$ returns a CT edge id where the edge $(v_1, v_2)$ belongs. Up($v$) and Down($v$) returns the parent and child vertex connected to $v$ respectively in ACT. Note that Up($v$) or Down($v$) may return more than one vertex if $v$ is a $join$ or a $split$. In such a case, the function, 'Up' or 'Down', is applied multiple times. We use an array $N^t[v]$ to store the node for the critical point where the contour class containing $v$ is created at time $t$. $ACT^t$ [6] can be constructed during the $CT^t$ construction.

STEP 2
Input : $ACT^t$ , $va$
Output : $JT_E^t$
1: **for** $i \leftarrow nv-1$ **to** 0
2: $v \leftarrow va[i]$;
3: **if** ($v$ is *upper leaf*) **then**
4: $n \leftarrow$ CreateNode($f^t(v)$,{E($v$,Down($v$))});
5: **if** ($v$ is $join$ or $split$) **then**
6: $n \leftarrow$ CreateNode($f^t(v)$,ESET($N^t$[Up($v$)]) $-$ {E($v$,Up($v$))} + {E($v$,Down($v$))});
7: Connect($n$,$N^t$[Up($v$)]);
8: **if** ($v$ is *lower leaf*) **then**
9: $n \leftarrow$ CreateNode($f^t(v)$,ESET($N^t$[Up($v$)])$-${E($v$,Up($v$))});
10: Connect($n$,$N^t$[Up($v$)]);
11: **if** ($v$ is $regular$) **then**
12: $N^t[v] \leftarrow N^t$[Up($v$)];
13: **else** $N^t[v] \leftarrow n$;

Step 2 computes a join tree where each edge $e^{t+1}$ is labelled with a set of $CT^t$ edge ids which corresponds to $e^{t+1}$. We process each vertex of $ACT^t$ in a decreasing order based on its function value. If $v$ is a regular vertex of $ACT^t$, no change occurs with respect to constructing $JT_E^t$. Otherwise, the topology of contours changes at $v$ when $w$ passes through $f^t(v)$ and ESET needs to be updated. Since $v$ is processed in a decreasing order, ESET of $N^t$[Up($v$)] is always computed and accessible before processing $v$.

The third step is to find correspondence to compute $JT_C^{t+1}$ from $JT_E^t$, which is the most essential part of the whole algorithm. Let's assume we have separate meshes $M^t$ for time $t = 1, 2, ..., T$ where vertex positions and connectivity of each mesh are exactly same. First, all the vertices in $M^t$ and $M^{t+1}$ are sorted in an increasing order based on the function value $f^t$ and $f^{t+1}$ defined on the ver-

tices. The sorted vertices are stored in an array $va2$. The processing is done vertex by vertex starting from the vertex which has the highest value. Assume that the region defined as $\{x|f(x) \geq f(v)\}$ is marked incrementally on the mesh $M^t$ and $M^{t+1}$ as each vertex $v$ is processed . The marked area is conceptually same as an *upper object*. Upper objects are updated for each iteration of the loop (line 1).

STEP 3
Input : $JT_E^t$ , $ACT^t$ , $ACT^{t+1}$, $va2$
Output : $JT_C^{t+1}$
1: **for** $i = 2vn - 1$ **to** 0
2: $(v, time) \leftarrow va2[i]$; // *time* = $t$ or $t+1$.
3: **if** collisions between object sets $X^t$ and $X^{t+1}$ occur when $w$ is decreased from $f(va[i-1])$ to $f(va[i])$ **then**
4: **for** each collision point $x_c$ between objects $X_k^t$ and $X_{k'}^{t+1}$
5: $lv_1 \leftarrow$ LowestVtx($X_k^t$);
6: $lv_2 \leftarrow$ LowestVtx($X_{k'}^{t+1}$);
7: $n \leftarrow$ CreateNode($f(x_c)$,ESET($N^t[lv_1]$)+ESET($N^{t+1}[lv_2]$));
8: Connect($n$,$N^{t+1}[lv_2]$);
9: $N^{t+1}[lv_2] \leftarrow n$;
10: ObjectUpdate($X_k^t$,$X_{k'}^{t+1}$);
11: **if** $time = t+1$ **then**
11: **if** ($v$ is *upper leaf*) **then**
12: $n \leftarrow$ CreateNode($f^{t+1}(v)$,ESET($N^t$[LowestVtx($X^t(v)$)]);
13: $N^{t+1}[v] \leftarrow n$;
14: **if** ($v$ is $join$) **then**
15: $n \leftarrow$ CreateNode($f^{t+1}(v)$,ESET($N^{t+1}$[Up($v$)]));
16: Connect($n$,$N^{t+1}$[Up($v$)]);
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: $n$ =CreateNode($f^{t+1}(v)$,NULL);
21: Connect($n$,$N^{t+1}$[Up($v$)]);
22: **else** $N^{t+1}[v] \leftarrow n$;
23: **if** $time = t$ **then**
24: **if** ($v$ is *upper leaf*) **then**
25: **if** $X^{t+1}(v)$ *exists* **then**
26: $n' \leftarrow N^{t+1}$[LowestVtx($X^{t+1}(v)$)];
27: $n$=CreateNode($f^t(v)$,ESET(E$^t$($v$,Down(v))+ESET($n'$);
28: Connect($n$ , $n'$);
29: N$^{t+1}$[LowestVtx($X^{t+1}(v)$)] $\leftarrow n$;
30: **else if** ($v$ is $join$ or $split$ or *lower leaf*) **then**
31: **for** each object $X_{k'}^{t+1}$ in time $t+1$
32: **if** $X_{k'}^{t+1}$ overlaps with $X_h^t(v)$ **then**
33: $w \leftarrow$ LowestVtx($X_{k'}^{t+1}$);
34: $n$=CreateNode($f^t(v)$, ESET($N^{t+1}[w]$) − ESET(E$^t$($v$,Up($v$))) + ESET(E$^t$($v$,Down($v$))));
35: Connect($n$,$N^{t+1}[w]$);
36: $N^{t+1}[w] \leftarrow n$;

As we take vertices $v$ in a decreasing order, objects increase their sizes. First, we need to check whether two objects $X_k^t$ and $X_{k'}^{t+1}$ collide at the point $x_c$ during the growth. Two objects already collided before are not considered for the test again. Since we use simplicial domain meshes, $x_c$ is always placed on an edge of the mesh (Figure 7 (a)). To detect the collision, we check the neighboring vertices of $v$ for each iteration of the loop (line 1). If a neighboring vertex $v'$ is covered and $v$ is not covered by an object from the other time, the collision point $x_c'$ on the edge $(v, v')$ and $f(x_c')$ can be computed using the values of $f^t(v)$, $f^t(v')$, $f^{t+1}(v)$ and $f^{t+1}(v')$. $x_c'$ and $f(x_c')$ are inserted to a priority queue ordered based on the value of $f(x_c')$. The queue is efficiently used for the collision test in line 3 of step 3.

The overlap of the two objects indicates that the contours on the border of the two objects have correspondence in the case of join tree. Therefore, a new node having $f(x_c)$ is created and the union

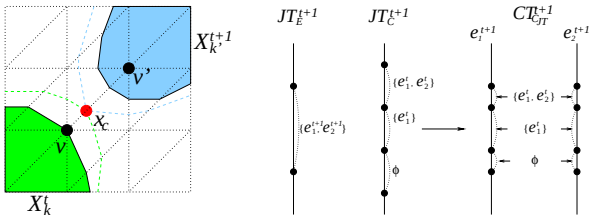



**Figure 7: (a) Collision Test. (b) Step 4.**

of ESETs in two objects are assigned as ESET of the node. In line 3, $f(v)$ means $f^{time}(v)$. LowestVtx($X_k^t$) returns the vertex with lowest value which is covered by an upper object $X_k^t$.

If $v$ is an *upper leaf* at $t+1$, a new node $n$ having $f^{t+1}(v)$ is created. If there is already an object $X^t(v)$ at $t$ in the place of $v$, the ESET of $X^t(v)$ is inserted into the node $n$. Otherwise, the ESET of $n$ becomes empty. If $v$ is a $join$ at $t = 2$, a new node $n$ having $f^{t+1}(v)$ is created. Since two or more objects are merged at $v$, the union of ESET for the objects is inserted into $n$ and connected to the nodes $N^{t+1}$ of merged objects. No change occurs in ESET computation For other cases in $t+1$,. If $v$ is an *upper leaf* at $t$, the $CT^t$ edge id of the newly created contour is inserted into ESET of the object $X^{t+1}(v)$ at $t+1$ covering the vertex $v$. If $v$ is a $join$ or $split$ or *lower leaf* at $t$, then find objects at $t+1$ which have an overlap with the object at time $t$ covering $v$ and update the ESET of the object at $t+1$.

The fourth step is to convert $JT_c^{i+1}$ into the form of the contour tree $CT_{c_{JT}}^{i+1}$ with correspondence information. The example of processing step 4 is presented in Figure 7 (b).

STEP 4
Input : $JT_C^{t+1}$, $JT_E^{t+1}$
Output : $CT_{C_{JT}}^{t+1}$
1: $ea \leftarrow$ sorted edges of $JT_E^{t+1}$
2: **for** $i =$ sizeof($ea$) $- 1$ **to** 0
3: $(v_a, v_b) \leftarrow$ two vertices of the edge $ea[i]$;
4: Subdivide $(v_a, v_b) \rightarrow (v_a, v_1)$,$(v_1, v_2)$,...,$(v_m, v_b)$ where $v_1, ..., v_m$ are vertices on the edges in $JT_C^{t+1}$ equivalent to $(v_a, v_b)$.
5: **for** each $e_k \in$ ESET($ea[i]$)
6: Insert the edges $(v_a, v_1)$,...,$(v_m, v_b)$ to $CT_{C_{JT}}^{t+1}$ in the place of $e_k$
7: ESET of the edges are taken from the $JT_C^{t+1}$.

The fifth step is to compute the intersection between ESETs of $CT_{c_{JT}}^{t+1}$ and $CT_{c_{ST}}^{t+1}$ to build $CT_C^{t+1}$. Each edge of $CT^{t+1}$ is corresponding to a set of edges with different ESET in $CT_{c_{JT}}^{t+1}$ and in $CT_{c_{ST}}^{t+1}$. We compare and compute intersection set of those edges and their ESET. This process is iterated for each edge of $CT^{t+1}$.

**Time Complexity Analysis** $n$, $m$ and $c^t$ are the numbers of vertices, tetrahedra, and critical points in $f^t$, respectively. This means the number of upper or lower objects at a certain isovalue can not be greater than $c^t$. The whole algorithm consists of five steps.

- Step 1 : $O(nlogn + m)$. This is analyzed in [6, 16].
- Step 2 : $O(n + (c^t)^2)$.
- Step 3 : $O(n + (c^t)^2 c^{t+1})$
- Step 4 : $O((c^{t+1})^2 c^t)$
- Step 5 : $O((c^{t+1})^2 (c^t)^2)$

In step 2, processing a critical point may take $O(c^t)$ for ESET computation, which makes the total complexity $O(n + (c^t)^2)$. In step 3, collision (line 3) 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 $t+1$. For each collision, we need to compute the union of ESET (line 7) with $O(c^t)$ time. The complexities for other parts of step 3 are minor. The total cost for step 3 is $O(n + (c^t)^2 \cdot c^{t+1})$. In step 4, the sorting process (line 1) takes $O(c^{t+1} log c^{t+1})$. The loop in line 2 has $O(c^{t+1})$ iterations. In line 5, the maximum number of $e_k$ is $c^{t+1}$. The line 6 may have $O(c^t)$ edges. Therefore, step 4 has $O((c^{t+1})^2 c^t)$. In step 5, we take intersections of edges and their ESET in $CT^{t+1}_{C_{JT}}$,$CT^{t+1}_{C_{ST}}$. As a result of the step 4, $CT^{t+1}_{C_{JT}}$,$CT^{t+1}_{C_{ST}}$ has at most $(c^{t+1})^2 c^t$ edges, which makes the total cost for the edge intersection $O((c^{t+1})^2 c^t)$. For each intersected edge, we need to perform ESET intersection which takes $O(c^t)$. The cost for step 5 is $O((c^{t+1})^2 (c^t)^2)$.

Processing steps 4 and 5 is generally too slow (see the timing results in section 9). Although $CT^{t+1}_C$ is the ideal form of holding the temporal correspondence information, all the information of $CT^{t+1}_C$ is embedded in $JT^{t+1}_C$ and $ST^{t+1}_C$ which is the output of the step 3. In run-time, every point $p^{t+1}$ in $CT^{t+1}_C$ can be easily mapped to points $p^t_{JT}$ and $p^t_{ST}$ in $JT^{t+1}_C$ and $ST^{t+1}_C$ respectively. The ESET of $CT^{t+1}_C$ at $p^{t+1}$ can be computed by intersecting the ESET of $p^t_{JT}$ and $p^t_{ST}$. In practice, we perform only the steps 1, 2 and 3 to save the preprocessing time and maintain $JT^{t+1}_C$ and $ST^{t+1}_C$ for run-time correspondence queries.

## 6. TOPOLOGY CHANGE GRAPH

The goal of this section is to classify topological events of time-varying isocontours and describe a run-time algorithm to construct a graph representing the topological events. When an isovalue $w$ is selected, isocontours for each timestep are defined as $I^t = \{C^t_1, ..., C^t_{n^t}\}$, $t = 1, ..., T$. 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. We define two contour mapping functions, $\psi$ and $\phi$. $\psi^t$ maps from a contour $C^t_k$ to a set of corresponding contours at $t+1$ which are evolved from the $C^t_k$. $\phi^t$ maps in an opposite direction from a contour $C^t_k$ to a set of corresponding contours at $t-1$ which evolves to $C^t_k$. Using these functions, we can define six topological events of time-varying isocontours.

- **create** : $C^t_k$ is created at time $t$ if $\phi^t(C^t_k) = \emptyset$.
- **disappear** : $C^t_k$ disappears at time $t+1$ if $\psi^t(C^t_k) = \emptyset$.
- **merge** : $C^{t-1}_{k_1}, ..., C^{t-1}_{k_m}$ are merged to form $C^t_k$ at time $t$ if $\phi^t(C^t_k) = \{C^{t-1}_{k_1}, ..., C^{t-1}_{k_m}\}$,$m \geq 2$.
- **split** : $C^t_k$ is split into $C^{t+1}_{k_1}, ..., C^{t+1}_{k_m}$ at time $t+1$ if $\psi^t(C^t_k) = \{C^{t+1}_{k_1}, ..., C^{t+1}_{k_m}\}$, $m \geq 2$.
- **continue** : $C^t_k$ continues at time $t+1$ if $\psi^t(C^t_k) = \{C^{t+1}_{k'}\}$ and $betti(C^t_k) = betti(C^{t+1}_{k'})$. $betti(C)$ is a betti number of a contour $C$, which can be pre-computed for all contours using [16].
- **genus change** : $C^t_k$ changes its topology at time $t+1$ if $\psi^t(C^t_k) = \{C^{t+1}_{k'}\}$ and $betti(C^t_k) \neq betti(C^{t+1}_{k'})$.

Now, the problem is to implement the function $\psi^t$ and $\phi^t$. We use correspondence information in time dependent contour trees to compute $\psi_t(C^t_k)$ and $\phi^t(C^t_k)$.

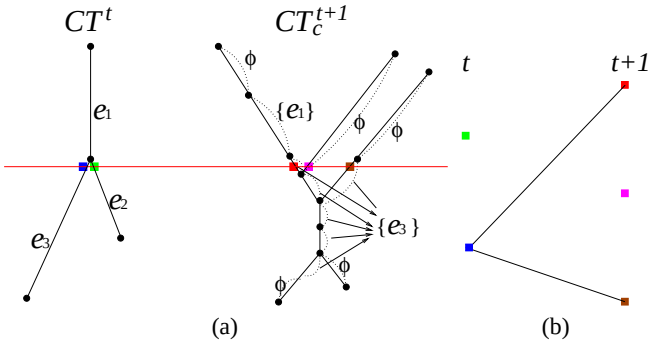



**Figure 8: (a) Checking Correspondence of Consecutive Contours. (b) Topology Change Graph**

Using the algorithms described in section 5 , we can construct the contour trees $CT^t$ and $CT^{t+1}_C$. When we select an isovalue $w_0$ in run-time, we can get the intersection points between contour trees $CT^t$ and $CT^{t+1}_C$, and a line $y = w_0$. Each intersection point in $CT^{t+1}_C$ represents a contour $C^{t+1}_k$. Since sets of corresponding edge, ESET, are already computed on every edge of $CT^{t+1}_C$, $C^{t+1}_k$ has an ESET $\{e^t_{k_1}, ..., e^t_{k_i}\}$, which uniquely defines $\{C^t_{k_1}, ..., C^t_{k_i}\}$ , $i = 0, 1, ...$ . If $i = 0$, $E^{t+1}_k = \emptyset$ and $\phi^{t+1}(C^{t+1}_k) = \emptyset$. By checking whether $i = 1$ and the $betti(C^{t+1}_k) \neq betti(C^t_{k_1})$, the genus change of a contour can be detected.

Those topological events can be visualized as a graph, called the *Topology Change Graph*(TCG). Each contour $C^t_k$ at time $t$ is represented as a node $N^t_k$ and a set of such nodes $S^t$ at time $t$ are laid vertically. A sequence of nodes $S^1$,...,$S^T$ are laid horizontally in time order. If $C^t_k \in \phi^{t+1}(C^{t+1}_{k'})$, two nodes $N^{t+1}_{k'}$ and $N^t_k$ are connected with an edge. This process is performed for all contours in each timestep sequentially to construct TCG. The function $\psi^t(C^t_k)$ can be implemented by checking every edge connecting $N^t_k$ and the nodes in time $t+1$. The betti number is stored in each node as a property and used to detect the genus change of a contour, which is not described in other feature tracking methods [12, 20].

## 7. QUANTIFICATION

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

Conventional algorithms [12,20] need to extract the surface first, and then the quantitative properties can be computed from the surface. However, such approaches are not suitable for interactive applications because surface extraction is expensive. Our approach pre-computes the quantitative properties of all possible contours for each timestep. This allows realtime evaluation of the properties for any selected contour in an interactive system.

Consider a 2D or 3D scalar field $f$ defined on a simplicial mesh. The length/area/volume of isocontours over the continuous range

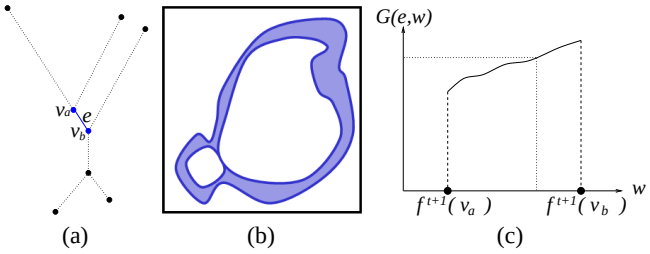



**Figure 9: Quantification for segmented contours. (a) $CT_C^{t+1}$. (b) Solid region corresponding to $e$. (c) Quantification Function for $e$.**

of isovalues in an $i$-th simplex can be represented with a univariate B-spline function $G_i(w)$. When such functions for all simplices are merged, a new single B-spline function $G(w) = \sum G_i(w)$ representing the quantitative properties of isocontours for any $w$ is constructed. If an isovalue $w_0$ is selected as an interactive parameter, the length/ area/ volume/ gradient of $I(w_0)$ is computed in real-time by evaluating the function $G(w_0)$. We refer to [1, 17] for detailed description of the $G(w)$ computation.

The main goal of this section is to compute $G(e, w)$ representing quantitative properties for each edge $e$ of a contour tree . Each edge $e$ in a contour tree corresponds to the region covered by a continuous contours. Let's assume the region is contained by a set of simplices $S_e = \{s_{(e,1)},...,s_{(e,n)}\}$. The function associated with the edge $e$ is the sum of the B-spline function for each simplex $s_{(e,k)}$ (Figure 9).

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

Once $G(e, w)$ is computed for each edge $e$ of the contour tree, the quantitative properties of any contour corresponding to a selected point with isovalue $w_0$ on $e$ can be quickly computed by evaluating $G(e, w_0)$ in run-time. Each node of TCG is colored based on the magnitude of the geometric property (Figure 10(c)).

In this paragraph, we describe an algorithm to compute $S_e$. Let $e = (v_1, v_2)$ such that $f(v_1) \leq f(v_2)$. We first pick any seed cell $c$ in the edge $e$ and use a propagation method similar to [2]. This takes $O(|S_e|)$.

```
Input : CT , path seeds [5] , e
Output : S_e
1: c ← a seed cell in an edge e of CT;
2: Enqueue c;
3: Visit(c);
4: while queue is not empty do
5:     s ← Dequeue();
6:     t ← 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 ← tetrahedron sharing the face t_i with s;
10:            if c is not visited then
11:                Enqueue(c);
12:                Visit(c);
```

The 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 $t$. The line 8 checks whether the triangle $t_i$ has overlap with the continuous contour class which is corresponding to the edge $e$. The Visit($c$) inserts the cell $c$ to $S_e$.

## 8. INTERACTIVE CONTOUR TRACKING

When time-varying isocontours have many evolving contours including noise, users may want to segment and visualize a subset of

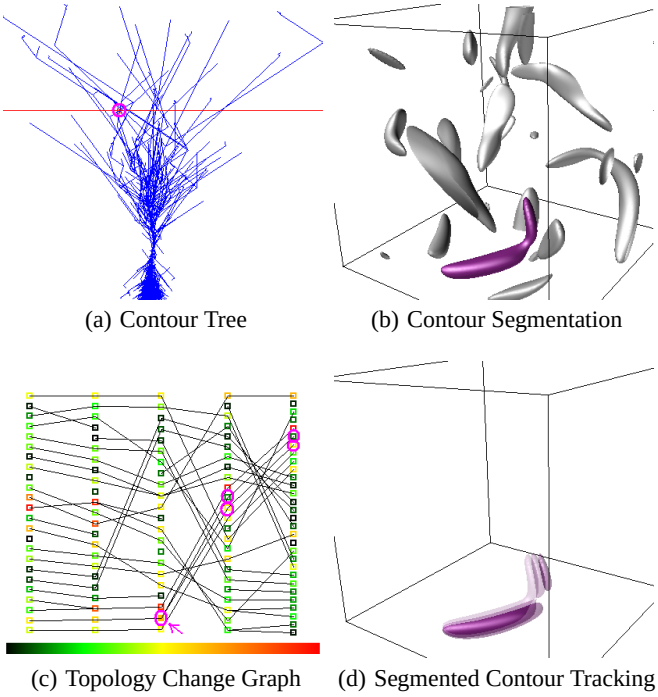



**Figure 10: Simulations of Turbulent Vortex Structures. The color of a node in (c) represents the surface area of a contour which corresponds to the node. The marked nodes correspond to segmented contours.**

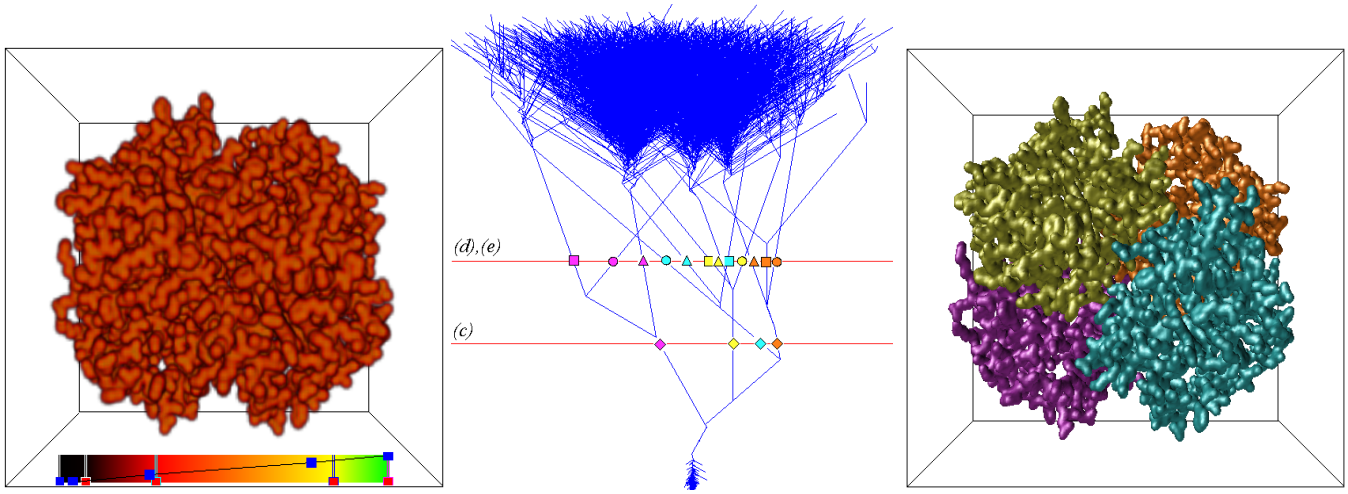
contours. This allows the user to focus on the evolution of interesting features. The topology change graph combined with quantitative information is used as an interface to guide identifying significant contours and their dynamic patterns. In this section, we describe an interactive algorithm to select and extract specific contours and track their evolution over time.

We use a seed set based isocontouring method for surface extraction [2]. A seed set $S$ is defined as a subset of cells such that any isocontour component intersects with at least one cell $c \in S$. Each contour surface can be constructed by mesh propagation from a seed cell containing the contour.

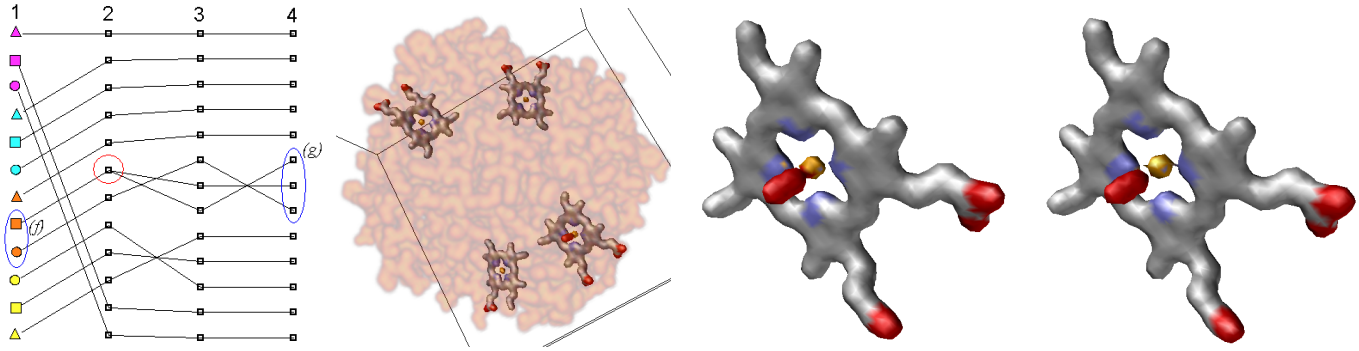
This process is efficient because visiting unnecessary cells, which has been considered as a main bottleneck of isocontouring, is avoided. Another important benefit of the seed set based isocontouring is its ability to segment a single connected component of an isocontour. Since we use a simplicial mesh, there can be at most one sheet of an isocontour in a cell. Starting from a seed cell, we incrementally track and triangulate the cells which are connected to the contour segment in the seed cell. Therefore, the surface generated by propagation from a seed cell is a single contour.

CT is useful for obtaining a minimal seed set and isolating individual contours. When a point on an edge of a CT is selected in run-time, a seed cell corresponding to the point is computed and the corresponding contour is quickly extracted by propagation from the seed cell. A detailed algorithm for computing a seed cell corresponding to a point on an edge of a contour tree is described in [5].

Suppose an isovalue $w_0$ is chosen and the TCG is computed. When a user selects a node $n^t$ at time $t$ in the graph, the corresponding point on the $CT^t$ is identified. A seed cell for this point is computed and the corresponding contour $C^t$ is extracted. The evolution of the selected contour is quickly tracked and displayed




(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

**Figure 11: Visualization of a hemoglobin molecule and its dynamics. In (c) each chain is segmented and colored. (f) and (g) shows a heme group composed of carbon (grey), 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).**

using the adjacency information of the graph. If $n^t$ is connected to the nodes $n_1^{t+1}$, ..., $n_k^{t+1}$ at time $t+1$, the corresponding contours of the nodes $n_1^{t+1}$ , ..., $n_k^{t+1}$ can be also quickly extracted in the same way as the extraction of $C^t$. The backward tracking can be done in the same manner. This process is shown in Figure 10.

## 9. RESULTS

We tested two time-varying data sets generated from hemoglobin molecular 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 undesirable artifacts [4] in the extracted surface. Visit our website [10] for additional details including pictures, movies and descriptions.

The first time dependent data set is an approximate electron density map of a deforming hemoglobin molecule (Figure 11(a)). Detailed description of the hemoglobin and its dynamics can be found in [11]. The hemoglobin is a protein that binds to oxygen in oxygen-rich areas (lung) and releases the oxygen to 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 Figure 11, the contour tree (b) of the density field (a) at time 1 indicates that the hemoglobin molecule consists of four (almost identical) polypeptide chains. A contour component of each chain is segmented and visualized with a different color using propagation from seed cells computed from the contour tree. Each chain has a flat ring structure called a *heme*, which is the active site in the oxy-deoxy process (Figure 11(e)). The rest of the polypeptide chain is called a globin. The contour tree (b) 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, which is shown in (Figure 11(d)). We can also track, quantify and visualize the evolution of each heme group (Figure 11(e)(f)(g).

The other data set is a pseudospectral simulation of coherent turbulent vortex structures [20] with a $128^3$ resolution and 33 time steps. Figure 10 shows the result of contour segmentation and tracking. When an isovalue 6.5 is selected, a TCG is constructed(c) where each node is colored with the surface area of a corresponding contour. A contour can be segmented and tracked over time interactively 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 consecutive functions (time 1 and 2) in an SGI ONYX

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

**Table 1: Timing results for $CT_C^{t+1}$ construction. (unit : sec)**

2 system with R12000 processors and 25GB main memory. The timing results are summarized in Table 1. The computation for each sequence of consecutive functions $(f^t, f^{t+1})$ can be done independently in a parallel system. The result shows step 4 and 5 are computationally expensive. As we mentioned in the time complexity analysis of section 5, all of the information in $CT_C^{t+1}$ are embedded in $JT_C^{t+1}/ST_C^{t+1}$ which is the output of the step 3. Therefore, in practice, we can just perform only step 1, 2, and 3 to generate $JT_C^{t+1}/ST_C^{t+1}$ for preprocessing. In run-time, when the correspondence information of a contour is requested, we can compute it from $JT_C^{t+1}/ST_C^{t+1}$.

The run-time operations are pretty fast. The construction of a topology change graph, quantification and tracking is completed in less than 0.1 sec for both data sets. Each of three contour surfaces in Figure 10 (d) are extracted in 0.28s, 0.28s and 0.29 where the number of triangles are 12025, 12652, and 12253, respectively. Generally, the contour extraction time scales linearly with the number of contour triangles.

## 10. CONCLUSION

We described an algorithm to compute correspondence information in time dependent contour trees. We use this information to extend all the benefits of a contour tree to the visualization of time-varying scalar fields. First, we extract a graph representing the topology changes of time-varying isocontours. Second, we segment, track, visualize and quantify the evolution of any selected contours. Finally, we accelerate extraction of a contour surface by generating the seed cells from each contour tree. The user interface which adopted above three features allows users to easily and quickly detect significant topological and geometric changes of time-varying isocontours.

## 11. ACKNOWLEDGEMENT

The authors are grateful to A. Thane for developing the CCV volume rendering tool ( Volume Rover ). We would like to thank Dr. David 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 UCSD 1018140 as part of NSF-NPACI, Interaction Environments Thrust, and a grant from Compaq for the 128 node PC cluster.

## 12. REFERENCES

[1] C. L. Bajaj, V. Pascucci, and D. R. Schikore. The contour spectrum. In *IEEE Visualization Conference*, pages 167–173, 1997.

[2] Chandrajit Bajaj, Valerio Pascucci, and Daniel R. Schikore. Fast isocontouring for improved interactivity. In *Preceedings of the 1996 Symposium for Volume Visualization*, pages 39–46, 1996.

[3] Praveen Bhaniramka, Rephael Wenger, and Roger Crawfis. Isosurfacing in higher dimensions. In *IEEE Proceedings of Visualization '2000*, pages 267–274, 2000.

[4] Hamish Carr, Torsten Möller, and Jack Snoeyink. Simplicial subdivisions and sampling artifacts. In *IEEE Visualization Conference*, pages 99–108, 2001.

[5] Hamish Carr and Jack Snoeyink. Path seeds and flexible isosurfaces using topology for exploratory visualization. In *Proceedings of VisSym*, pages 49–58, 2003.

[6] Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. *Computational Geometry: Theory and Applications*, 24(2):75–94, 2003.

[7] Yi-Jen Chiang. Out-of-core isosurface extraction of time-varying fields over irregular grids. In *IEEE Visualization Conference*, pages 217–224, 2003.

[8] Yi-Jen Chiang, Tobias Lenz, Xiang Lu, and Günter Rote. Simple and optimal output-sensitive construction of contour trees using monotone paths, 2003. `http://cis.poly.edu/chiang/contour.pdf`.

[9] Yi-Jen Chiang and Xiang Lu. Progressive simplification of tetrahedral meshes preserving all isosurface topologies. In *Eurographics '03*, pages 493–504, 2003.

[10] Website for Time-Varying Contour Topology. `http://www.ices.utexas.edu/~bongbong/time_analysis`.

[11] David Goodsell. Hemoglobin : Cooperation makes it easier. `http://www.scripps.edu/pub/goodsell/pdb/pdb41/pdb41_2.html`.

[12] G. Ji, H.-W. Shen, and R. Wenger. Volume tracking using higher dimensional isocontouring. In *IEEE Visualization Conference*, pages 209–216, 2003.

[13] Lutz Kettner, Jarek Rossignac, and Jack Snoeyink. The safari interface for visualizing time-dependent volume data using iso-surfaces and contour spectra. *Computational Geometry : Theory and Applications*, 25(1-2):97–116, 2003.

[14] W. S. Koegler. Case study: application of feature tracking to analysis of autoignition simulation data. In *IEEE Visualization Conference*, pages 461–464, 2001.

[15] V. Pascucci. On the topology of the level sets of a scalar field. In *12th Canadian Conference on Computational Geometry*, pages 141–144, 2001.

[16] V. Pascucci and K. Cole-McLaughlin. Efficient computation of the topology of level sets. In *IEEE Visualization Conference*, pages 187–194, 2002.

[17] Valerio Pascucci. Multi-dimensional and multi-resolution geometric data-structures for scientific visualization. Technical report, Ph.D. Thesis. Department of Computer Sciences, Purdue University, 2000.

[18] Han-Wei Shen. Isosurface extraction in time-varying fields using a temporal hierarchical index tree. In *IEEE Visualization '98*, pages 159–166, 1998.

[19] D. Silver. Object-oriented visualization. In *IEEE Computer Graphics and Applications*, pages 54–62, 1995.

[20] D. Silver and X. Wang. Tracking and visualization turbulent 3d features. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):129–141, 1997.

[21] J. K. Sircar and J. A. Cerbrian. Application of image processing techniques to the automated labelling of raster digitized contours. In *Int. Symp. on Spatial Data Handling*, pages 171–184, 1986.

[22] Philip M. Sutton and Charles D. Hansen. Isosurface extraction in time-varying fields using a temporal branch-on-need tree (T-BON). In David Ebert, Markus Gross, and Bernd Hamann, editors, *IEEE Visualization '99*, pages 147–154, San Francisco, 1999.

[23] S. Takahashi, T. Ikeda, Y. Shinagawa, T. L. Kunii, and M. Ueda. Algorithms for extracting correct critical points and constructing topological graphs from discrete geographical elevation data. *Computer Graphics Forum*, 14(3):C–181–C–192, 1995.

[24] Sergey P. Tarasov and Michael N. Vyalyi. Construction of contour trees in 3d in o(nlogn) steps. In *ACM Symposium on Computational Geometry*, pages 68–75, 1998.

[25] Marc J. van Kreveld, Rene van Oostrum, Chandrajit L. Bajaj, Valerio Pascucci, and Daniel Schikore. Contour trees and small seed sets for isosurface traversal. In *ACM Symposium on Computational Geometry*, pages 212–220, 1997.

[26] Chris Weigle and David C. Banks. Extracting iso-valued features in 4-dimensional scalar fields. In *IEEE Symposium on Volume Visualization*, pages 103–110, 1998.