Interactive Exploration of Large-Scale Time-Varying Data using Dynamic Tracking Graphs

Wathsala Widanagamaachchi*
SCI Institute, University of Utah

Cameron Christensen†
SCI Institute, University of Utah

Peer-Timo Bremer‡
SCI Institute, University of Utah
Lawrence Livermore National Laboratory

Valerio Pascucci§
SCI Institute, University of Utah

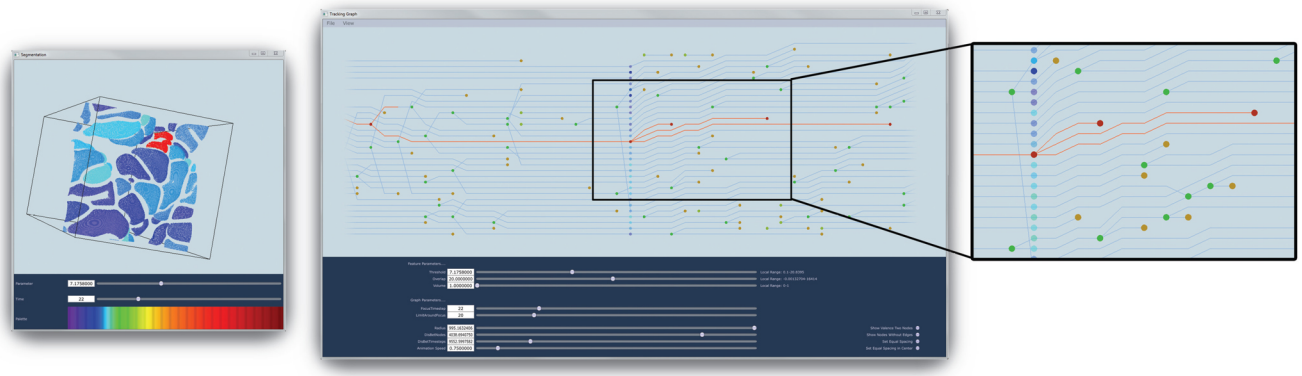

Figure 1: Our framework provides a natural and interactive exploration of tracking graphs (middle), combined with a traditional 3D view of features (left), and allows a cross-linked selection among them to enable a full spatio-temporal exploration of data. The selected feature is displayed in 'red' in the 3D feature viewer while the relevant subtree in the tracking graph is highlighted in the other viewer. The images displayed here use the H_Control_none dataset.

Abstract

Exploring and analyzing the temporal evolution of features in large-scale time-varying datasets is a common problem in many areas of science and engineering. One natural representation of such data is tracking graphs, i.e., constrained graph layouts that use one spatial dimension to indicate time and show the "tracks" of each feature as it evolves, merges or disappears. However, for practical data sets creating the corresponding optimal graph layouts that minimize the number of intersections can take hours to compute with existing techniques. Furthermore, the resulting graphs are often unmanageably large and complex even with an ideal layout. Finally, due to the cost of the layout, changing the feature definition, e.g. by changing an iso-value, or analyzing properly adjusted sub-graphs is infeasible.

To address these challenges, this paper presents a new framework that couples hierarchical feature definitions with progressive graph layout algorithms to provide an interactive exploration of dynamically constructed tracking graphs. Our system enables users to change feature definitions on-the-fly and filter features using arbitrary attributes while providing an interactive view of the resulting tracking graphs. Furthermore, the graph display is integrated into a linked view system that provides a traditional 3D view of the current set of features and allows a cross-linked selection to enable a fully flexible spatio-temporal exploration of data. We demonstrate the utility of our approach with several large-scale scientific simulations from combustion science.

Keywords: Time-Varying Data, Feature Detection and Tracking, Parallel Coordinates, Topology-based Techniques, Visualization in Physical Sciences and Engineering.

1 Introduction

Defining, extracting and anlyzing features in the increasingly large and complex datasets produced by state of the art simulations still poses a significant challenge. Coupling the corresponding analysis across time steps to understand the temporal evolution of such features increases the difficulty exponentially. First, a temporal analysis multiplies the amount of data that must be considered simultaneously, often exceeding available memory and other resources. Second, the resulting data potentially contain information about thousands of features across hundreds of time steps making it challenging to present them in a comprehensible manner.

Conceptually, the most natural representation of such data is a *tracking graph* showing the evolution of all features across time as a collection of feature *tracks* that may merge or split. Given some feature definition a graph is constructed by first extracting all features from all time steps and then correlating those features across time, for example, by considering their spatial overlap. For the terabyte-scale simulations common today, this step often results in hours or days of file I/O time alone, making dynamic modification of feature parameters or correlation criteria infeasible. Furthermore, creating a suitable graph layout for the entire time series may take hours using existing tools preventing any interactive changes.

*e-mail: wathsy@sci.utah.edu
†e-mail: cam@sci.utah.edu
‡e-mail: bremer5@llnl.gov
§e-mail: pascucci@sci.utah.edu

IEEE Symposium on Large Data Analysis and Visualization 2012
October 14 - 15, Seattle, Washington, USA

Finally, for all but the smallest datasets, such graphs, even assuming an optimal layout, quickly become too large and convoluted for users to understand. Consequently, most existing techniques to visualize and explore time-varying datasets have primarily focused on high dimensional projections [47], illustration [23], and change detection [36].

However, recent advances in topological analysis have resulted in techniques that are able to efficiently extract and encode entire feature families in a single analysis pass [3, 4]. Instead of computing a set of features per time step and parameter setting, these approaches compute a meta-representation that encodes all possible features for a wide range of parameter settings. This reduces the problem of feature computation per time step to a single pre-processing step easily performed in parallel. Nevertheless, the problems of feature correlation, graph layout and display still remain.

In this paper we address these challenges by creating a new flexible, efficient, compact meta-graph which, similar to the topological encoding for features, stores not one particular tracking graph but instead the entire familiy of graphs for all possible feature parameters. From this meta-graph we can interactively extract a specific graph for particular parameters and correlation metrics. We have also developed progressive layout algorithms that allow us to interactively visualize the resulting graph, providing the user freedom to change both feature parameters and correlation metrics on the fly. In particular, starting from a time step of interest, we progressively lay out the tracking graph both forward and backward in time. The user can interactively change timesteps or expand and contract the window of interest and thus explore the entire graph. In fact, this approach often results in layouts that are locally superior to a global approach as fewer constraints must be considered. Furthermore, we provide several techniques to filter and/or simplify the graph based on various feature attributes in order to reduce its visual complexity. The graph visualization is tightly integrated with a traditional three-dimensional visualization of the features allowing for the first time a fully interactive exploration of spatio-temporal features in massive data sets. Our contributions in detail are:

1. Define and construct a meta-graph in pre-processing from a sequence of feature families;
2. A progressive algorithm to extract, filter, and simplify a tracking graph from a meta-graph;
3. A progressive two-stage layout algorithm for tracking graphs; and
4. An interactive linked view system combining tracking graphs and feature displays.

We have tested and validated our framework with several large-scale scientific simulations from combustion science. However, the implementation of our framework is general in such a way that it can be applied to other scientific domains where feature-based analysis is relevant.

2 Related Work

In this section, we highlight some of the past work involved in this domain. To facilitate better understanding, we analyze these related work by categorizing them into four sections: Feature Extraction, Feature Tracking, Time-Varying Data Visualization and Layered Crossing Minimization.

Feature Extraction: In using feature based approaches for the analysis of scientific data, techniques such as isosurfaces [30], interval volumes [17] and thresholding combinations of various scalar quantities [41] have been used before. However, any changes to the attributes associated with these features requires re-processing of the entire time-dependent simulation, so these techniques are costly to apply. Thus, the use of topological techniques have become increasingly popular, in which the concepts of Morse theory [33] are used to identify similar sets of features. The well-developed notion of simplification provided by Morse theory makes the resulting feature analysis techniques hierarchical. In these topological techniques, Reeb graph [32], contour tree [1, 7, 44] and Morse-Smale complex [21, 28] have been used to define and extract features. The contour tree encodes topological features for all isovalues of a function. We base our segmentation method on the merge tree (also called a join or split tree) which is a sub-structure of contour trees that tracks either merges or splits of the isocontours [6].

Feature Tracking: Defining and tracking features of interest has long been an area of interest in the visualization community. It can identify correspondences between features and reveal temporal trends in the underlying time-varying data. Methods which are used for defining, matching and tracking these features can vary based on the feature type. Data features can be matched based on their corresponding positions [38] while topological features can be tracked using high dimensional geometries [4, 3]. As such [19, 40, 16, 18], have used critical points in geometric models for feature tracking.

In another line of research, feature attributes like position and size have also been used for measuring data changes and identifying feature matches. Samtaney et al. [36] apply methods from object tracking in image processing to feature tracking. In doing so, they make use of feature attributes such as centroid, volume and moment between time steps. Volume overlap of features has also been used to identify these correspondences [38, 39, 28] and we too utilize this method to base the feature tracking step in our approach. In some follow-up-work, motion prediction has also been used to improve the feature matching accuracy [34].

Time-Varying Data Visualization: Due to the dynamic nature of time-varying data, general data visualizations differ considerably from time-varying data visualizations. Traditionally, snapshots of individual timesteps or animations have been used for visualizing the evolution of data. Other techniques like High dimensional projection [47], Illustration [23] and Change detection [36] have also been used.

Most current time-varying data visualizations focus on managing data more efficiently either by using optimized data structures [43, 46] or by using compression techniques [37]. The recent advances in graphics hardware have also facilitated the production of interactive visualizations [31]. As such, with the use of hardware accelerations, Wavelet and Moving Picture Experts Group (MPEG) compression have been applied to time-varying data for achieving real-time decompression and interactive playback [20].

Previous work like [36] and [34] are closely aligned with our research, since they also utilize tracking graphs to show the evolution of features. Reinders et al. [34] employed a linked-view interface similar to ours to assist the visualization. However, our framework has the additional ability of handling large-scale time-varying datasets.

Layered Crossing Minimization: Crossing minimization is a problem which has been extensively studied in related areas. Most crossing minimization techniques require the graphs to be in a hierarchical manner and operate on it in layers, where the vertices in each layer of the graph are permuted to minimize the total number of crossings.

Barycenter [42, 29], Median [15], Greedy-insert [14], Greedy-switch [14] and Sifting [35] provide examples for some well known heuristics for theS 2-layer crossing minimization problem. Although heuristic-based algorithms are simple to implement and offer speedy execution at run-time, they are inefficient in several other aspects. Most heuristic-based techniques use a layer-by-layer-sweep to reduce crossings in a layered graph. That is, they push crossings downwards or upwards in the graph until they are resolved at layer k. Since these algorithms are restricted to a local

view, they can get stuck in a local minimum, leading to sub-optimal solutions.

Apart from heuristic-based techniques, several exact methods have also been used as solutions to this problem. Subdivision-based formulation, ordering-based formulation and branch-and-cut-and-price are the three main types of exact methods. The two widely known exact methods are integer linear programming (ILP) [11, 25, 24] and semidefinite programming (SDP) [5, 10]. The SDP formulation is used mostly on denser graphs while ILP is usually faster on sparse ones. In addition to these, methods based on Planarization [8, 9] and Adjacency matrix transformations [27] have also been studied.

Due to the progressive nature of our system, we opted to use the median heuristic, a technique that uses a layer-by-layer sweep.

3 Topological Feature Families

Our goal is to interactively explore and track the evolution of features in a time-varying dataset. One of the most important degrees of freedom of such a system is the ability to quickly change feature parameters, for example, by adjusting thresholds or simplification levels. However, given the expected data sizes, on-the-fly feature computation is practically infeasible and would require massively parallel computing resources. The alternative is to pre-compute features for a wide range of potential parameters and to store the results in an efficient look-up structure. Here we are using topological feature families [3, 2] which can encode a wide range of feature types and simplifications using hierarchical merge trees. This section briefly introduces the necessary concepts of topology and describes how a particular set of features and their attributes can be efficiently exracted from the feature family represented by each merge tree.

3.1 Hierarchical Merge Trees

For a smooth simply connected manifold M and a function $f : M \to R$, the *level set* of f at the isovalue s, $L(s)$ is defined as the collection of all points in M with the function value $s : L(s) = \{p \in M | f(p) = s\}$. A *contour* is defined as a connected component of a level set. The merging of contours as the isovalue s is swept from top-to-bottom through the full range of f can be represented by a *merge tree* (Figure 2(a)-(e)). A leaf appears in the merge tree each time a new contour appears. That is, each time the isovalue passes through a maximum, a new leaf appears in the merge tree. Each branch in the merge tree represents a neighboring set of contours which are subsets of M and joining of branches in the tree indicates contour merging.

Merge trees are ideally suited to encode threshold-based features, e.g. regions around maxima or minima [28, 32, 4, 3, 2]. For example, given a value t within the full range of f, the corresponding threshold based features can be found by "cutting" the merge tree of f at t. This creates a forest of sub-trees, where each sub-tree represents a connected component existing at t. Note that this cut is not necessarily a horizontal line. For example, Laney et al. [28] use a simplification-based feature definition using *persistences* (the length of leaf branches) to create a cut through the graph and Mascarenhas et al. [32] use a locally scaled version of the same metric called *relevance*.

As indicated in the figures there exists a natural correspondance of branches in the merge tree to regions of space. Storing this segmentation information allows one to easily construct the geometry of a subtree/feature as a union of branch segmentations. Furthermore, one can precompute feature attributes such as first order statistical moments or shape characteristcs for each feature which are also stored on a per-branch basis. While merge trees are highly efficient and flexible, they quantize the space of features to those involving function values of critical points. This implicit quantization is often too coarse to be practical. In this case we compute

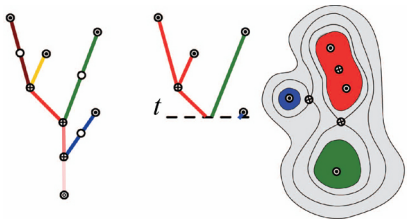

Figure 3: Segmentation for a particular function value, t: obtained by cutting the merge tree at t and ignoring all pieces below. Each subtree in the forest of sub-trees is considered as a single feature.

an *augmented contour tree* by introducing additional valence-two nodes to split the branches which are longer than a certain desired interval [6] (Figure 2(f)). In this manner, hierarchical merge trees encode an entire *feature family* alongside its segmentation and relevant attributes in a compact and efficient manner. We use the algorithm proposed in [3] to construct feature families for all time steps in the pre-processing step.

Starting from a sequence of feature families our framework consists of two stages: an offline pre-processing step to compute a meta-graph storing all possible tracking graphs, and a linked view visualization to interactively explore the spatio-temporal feature space.

4 Meta-Graphs

As discussed above, storing feature families removes the need for repeated feature computation and thus makes interactive feature selection possible. However, creating a tracking graph from a set of features remains an expensive operation. For each consecutive pair of time steps one must correlate all features. This typically requires multiple traversals of the corresponding segmentations as well as constructing search structures. Given hundreds of time steps this cannot be done interactively. Instead, we propose a new structure called a *meta-graph* that, similar to the merge tree for features, encodes all possible tracking graphs and relevant attributes. In this section, we describe how to create the meta-graph between two feature families, how to augment it with various feature attributes and correlation metrics, how to progressively extract a tracking graph, and finally how to filter and/or simplify this graph in preparation for layout and display.

4.1 Meta-Graph Computation

The purpose of the meta-graph is to encode, for all possible features at time T, their correlated features in time steps $T-1$ and $T+1$. The key observation is to efficiently compute and encode this information so that feature families are nested independent of the exact feature definition, e.g. threshold, persistence, relevance, etc. More specifically, since features are represented by subtrees, each feature that is correlated with a branch b is also correlated with all its parent branches. In other words, to use the metaphor of terrain, the top of the mountain is correllated with its base. We use this fact judiciously to create a simple yet efficient algorithm to compute meta-graphs.

The algorithm proceeds in two steps. First, we create correspondences between branches of consecutive merge trees. Second, we accumulate the resulting information for all subtrees/features. For the first step, we load both segmentation of the two corresponding trees. Note that these segmentations are typically stored in a sparse format and do not cover the entire domain. We therefore construct a look-up of vertex ids to segment/feature ids for one of the segmentations. Subsequently, we iterate over all vertices of the other segmentation and determine whether it overlaps with some segmented vertex of the first and if so compute the corresponding segmentation

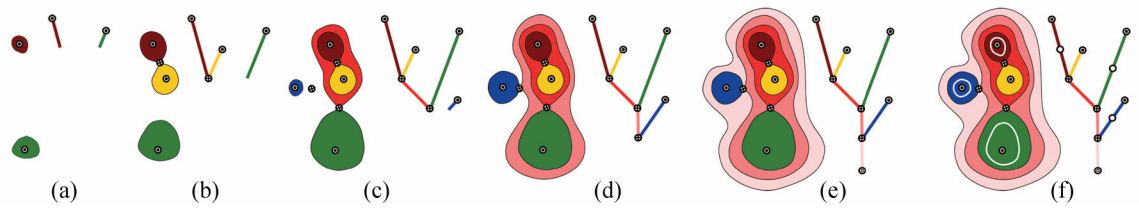

Figure 2: (a)-(e) Merge tree construction by recording the merging of contours as the function value is swept top-to-bottom through the full value range. (f) Augmented merge tree: obtained by splitting all branches in the merge tree which span more than a given range (see Figure 2 in [3]).

id. For each successful pair of overlapping segments $(\sigma_i^T, \sigma_j^{T+1})$ in time steps $T, T+1$ we either: add an edge to the meta-graph; or if an edge already exists, increase a counter, tracking the amount of *overlap* between both segments. See Figure 4.

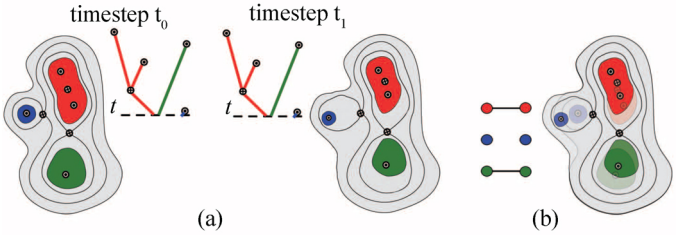

Figure 4: Initial edges of the meta-graph for two time steps. (a) Feature segmentations at t for both time steps (b) Obtaining the physical overlap between each of the two features in segmentation space to determine the existence and overlap values of edges in the meta-graph.

As mentioned above, each subtree of the merge tree represents a feature. In the second step, we accumulate both the edges and the overlap information according to the nesting relationships in the hierarchy, associating each feature to the root branch/node of its corresponding subtree. In this manner, each feature has a clearly defined *lifetime*. In the case of a merge tree, a feature is *born* at the parameter value of the upper vertex of the branch, it *merges* for parameter values below its root, and it is *alive* for parameters in between its two nodes. For each edge (σ_i, σ_j) between nodes, we determine whether the lifetime of the corresponding features overlap or not. In the second case, we find the ancestor(s) of the feature with the higher lifetime, say $\Sigma_k = ancestor_{\sigma_j}(\sigma_i)$ whose lifetimes do intersect. We then remove the edge (σ_i, σ_j) and instead add the corresponding edges for all ancestors in Σ_k to σ_j.

The resulting meta-graph contains nodes corresponding to all features and each node contains edges to features in adjacent time steps whose lifetimes overlap. In practice, we store the graph as a collection of individual files, using one file per consecutive feature family pair.

4.2 Dynamic Tracking Graphs

Given a meta-graph and the corresponding feature families one can easily extract a tracking graph for a particular parameter setting. Given a parameter t, we query the feature families for all nodes alive at this value and extract all edges between two living nodes. In practice, we maintain an active set of living nodes that is dynamically updated by including the parent or child branches of nodes as the threshold changes. Additionally, we provide the ability to filter edges according to the amount of overlap which effectively changes the correlation criterion on the fly. Note that one could easily use more advanced correlation criteria and store the relevant information on the graph edges.

As discussed in more detail in Section 5, one is rarely interested in the entire graph as it is typically too complex to be comprehensible. Instead, we exploit the fact that the graph is stored in individual time step "slabs" and, starting for a user-selected *focus* time step, T_i, create the graph progressively by first adding the nodes/edges of T_{i+1} then of T_{i-1}, of T_{i+2}, etc. At this point we also filter and/or simplify the graph. For example, we typically suppress nodes with no edges as they represent noise and allow the user to subselect based on various feature attributes, e.g. their volume and/or other statistics. If a node is filtered all incident edges are removed as well and nodes left without edges are removed recursively. As indicated in Figure 7 the entire pipeline is implemented in an asynchronous streaming fashion. Slabs are read from disk in order of distance, adapted to the current parameter, passed through the filter module, and handed off to the layout modules.

5 Graph Layout and Visualization

As discussed above, our framework is designed specifically to remain interactive even for large graphs and thus we focus on progressive techniques to both layout and visualize tracking graphs. Consequently, we always process graphs with respect to a focus time step selected by the user. Starting from this time step we iteratively add nodes and edges both forward and backward in time up to a user-defined time window. Furthermore, computing optimal or near-optimal layouts may be expensive for larger graphs and thus we use two different strategies. As slabs of the graph are read and filtered from the first two modules, we start both a fast greedy layout and a slower more optimized one. As a result, the user is immediately presented with a suboptimal layout which is replaced with a better one as soon as it is available.

5.1 Greedy Layout

To quickly create a reasonable layout we exploit the fact that feature families are represented by (binary) trees. As a result, within each timestep one layout is given by the depth first ordering of features placing the root in the middle and its subtrees recursively on either side. Note that in this step we simultaneously "place" all features independently of their lifetime. While this appears to be a random and clearly not optimal choice the layout is better than expected. The reason is that all trees come from a continuously evolving simulation and are constructed by the same code processing the data in the same order. Therefore, the trees naturally retain some temporal coherence making the depth first layout a good initial choice. Furthermore, since the structure of the tree does not change, this layout needs to be computed only once. In practice, we store the resulting order and simply adjust the spacing of the given set of living nodes to be equal. A typical greedy layout is shown in Figure 5(a).

5.2 Optimal Layout

As the system progressively places and displays time steps in the greedy layout, a background process works on computing an optimal layout. While it is possible to use global layout techniques on all currently loaded time steps this would result in an algorithm

(a) (b)

Figure 5: Graph layouts produced by our framework using the H_Control_none dataset. (focus timestep=6, threshold=15, overlap=20, volume=1.0) (a) Greedy layout produced as a result of ordering all features in each timestep in a depth first manner (b) Optimal layout produced using a layered optimization scheme, with the use of the median heuristic.

of order $O(T^2)$ as the first three, then the first five, first seven, etc. timesteps are considered. Instead we have chosen a layered optimization scheme moving outward from the focus time step. In particular, we assume the layout of the "inward" time step to be fixed and use the median heuristic to place new nodes. The median heuristic places each node in the next time step at the median of all nodes it is connected to in the current time step. This optimization is aimed at minimizing edge length and thus reduces crossings. Such heuristics are known to work well for shorter sequences which typically are far less constrained than a global tracking graph. As shown in Figure 5(b), it is quite common for our system to return layouts of smaller graphs entirely free of intersections. Another important aspect is the placement of newly created nodes not connected to the current time step. One strategy used, for example by the popular dot system [26], is to place such nodes on the top (or bottom) of the graph. However, this often results in graphs with extreme aspect ratios as shown in Figure 6(b). Instead, we maintain a list of "empty" positions left by the median heuristic in which to place new nodes. This results in much more compact and visually appealing layouts. See Figure 6(a) & 6(c).

5.3 Local Layout Adaptation

Once an optimal layout is computed for a time step, we adapt, rather than recompute this layout in case of parameter changes. As the user selects new filter values, parameters or overlaps, nodes and edges can appear, disappear, merge or split. Instead of reverting to the greedy layout which often would be a somewhat drastic change, we remove and add nodes to the current layout and simply adjust the scaling along the y-axis. While not optimal, this strategy preserves temporal coherence and typically results in a significantly better layout than the default greedy one. Furthermore, using animation for splitting or merging of nodes conveys important structural information to the user. Nevertheless, any change in the structure of the graph triggers a new optimal layout and thus ultimately this is what the graph will converge to.

5.4 Graph Visualization

In visualizing the tracking graph, our system uses several visual effects like animation, fading, feathering and correlated color mapping to help the user maintain context. Any time the layout of a time step changes we animate the nodes to slowly approach their new position over several frames. This preserves the context and prevents rapid and drastic changes in the layout. Furthermore, we allow to hide nodes of valence two to prevent visual clutter and highlight important events such as births, deaths, splits, and merges. Similar to the animation approach, nodes and edges that appear or disappear are blended in or out to prevent visual artifacts. Finally, edges leaving the time window of interest are feathered out.

Nodes can be colored using various attributes such as their volume or the function value of their highest maximum. More importantly, the colormap between the segmentation and the graph display is shared providing an important link and contextual information to the user. For more details we refer the reader to the accompanying video.

6 System Details and Implementation

In this section we describe different aspects of our system, the implementation and various design choices made to ensure interactivity. The system is implemented using the ViSUS framework [45] which provides the basic building blocks for designing a streaming, asynchronous dataflow. As usual the dataflow is built up from different nodes connected by pipes that pass messages and data.

The core of our dataflow is shown in Figure 7. The Data Reader continuously checks whether all requested data has been loaded and if not passes the corresponding file(s) one-by-one immediately into the filter module. The reader will cache slabs of the tracking graph as long as there is memory available. The filter first extracts the relevant nodes and edges according to the current parameter and then proceeds to filter according to attribute values, valences, correlation metric, etc.. The reduced slab is then sent simultaneously to the two graph layout modules that will compute the corresponding layout. Once a layout has been computed it is passed on to the rendering module which integrates it with the currently drawn graph. Each time the user changes relevant parameters or subselections the current processing is interrupted and restarted, though the renderer maintains its current state for visual continuity.

Since individual feature families tend to be small (compared to the raw data) the first couple of time steps are processed rapidly providing instantaneous feedback to the user. Subsequently, the layout is adjusted in a progressive and continuous fashion allowing the user to judge how long a given graph will likely require for an optimal layout. In our examples the graph drawing is typically not restricted by the performance of the system but rather by the complexity of the graph. For large sets of features or broad windows of time the system will quickly create layouts for a larger number of

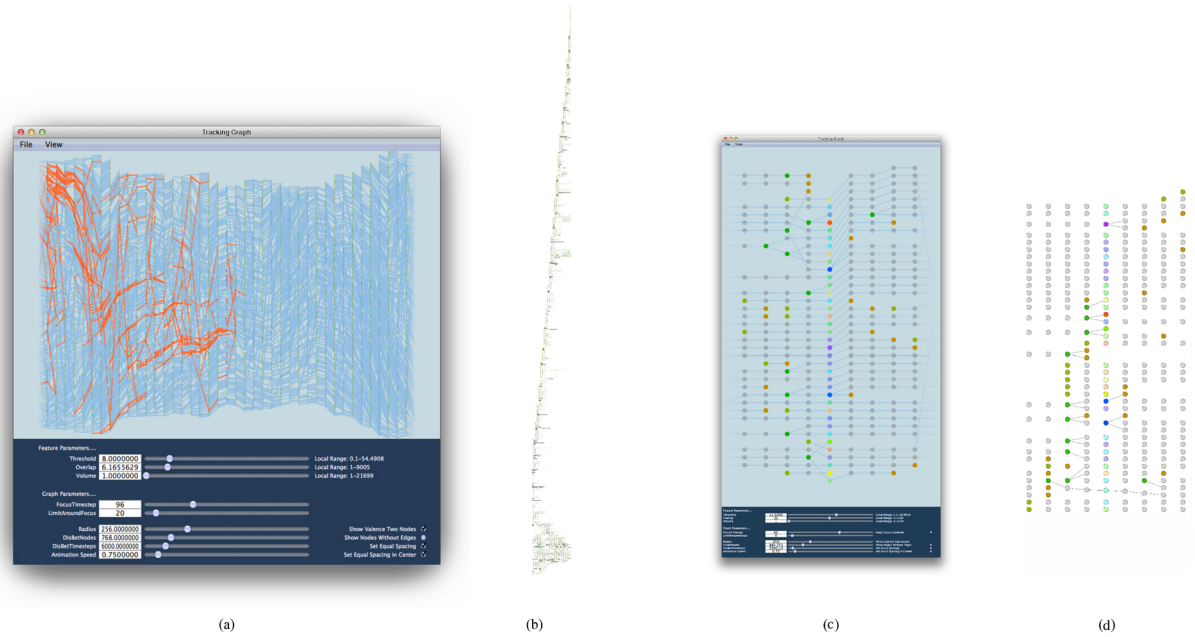

Figure 6: For a complex tracking graph, from the Swirl dataset, (a) & (b) compares the optimal layout produced by our framework with the one produced by Dot, respectively. Similarly, (c) & (d) compares the optimal layout produced by our framework with the one produced by Dot, for a much simpler tracking graph, from the H_Control_none dataset.

nodes than is feasible to comprehend easily. Usually this results in a further subselection by the user.

6.1 User Interface and Interaction

The user interface consists of two main views: one for the tracking graph (Figure 8(b)) and one for the 3D segmentation (Figure 8(a)).

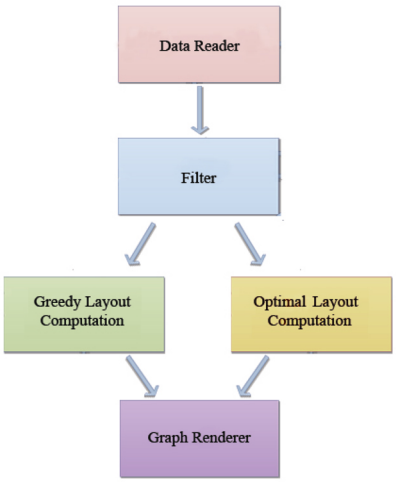

Figure 7: The core of the dataflow within the tracking graph viewer. Data Reader module reads the necessary data and passes them onto the filter module where the relevant nodes and edges are extracted according to the current parameters and attributes. The resultant data are sent to the two graph layout modules simultaneously to compute the corresponding layouts. Graph Renderer module uses the data received from the graph layout modules to display the graph.

Both windows allow the user to select the primary feature parameter and focus time step and the selections are linked. In addition, the tracking graph display provides various options to filter nodes based on volume, edges, overlap and choosen number of time steps. We also allow the user to hide valence-two nodes and instead display unbroken lines to save screen space. Both windows allow the cross-linked picking of features and the graph display will automatically highlight the subgraph containing the selected feature as shown in Figure 1. Finally, we provide the option to color nodes based on various attributes such as their volume, the highest maximum present in the subtree etc.

7 Results

We demonstrate the power of our technique using three large-scale combustion simulations. The H_control_none dataset simulates an idealized premixed hydrogen flame with no turbulence. Here we are analyzing 100 time steps of an adaptive mesh resolution (AMR) simulation at an effective resolution of $256 \times 256 \times 768$ totaling about 400GB of input data [12]. The SwirlH2 dataset represents a device scale, lean, premixed, low-swirl flame computed using a low Mach number combustion code LMC [13]. The data consists of 331 time steps of an AMR grid at an effective resolution of $1024 \times 1024 \times 1024$ totaling about 4 TB of raw simulation data. In both data sets the features of interest are burning cells defined as regions of high fuel consumption. However, there exists no unique threshold and exploring the evolution of burning cells under varying thresholds is one of the primary goals of this analysis. Here we compute feature families of the fuel consumption rate as well as various feature-based statistics such as the volume of features, their average temperature, and their position. Unlike in previous studies in [4, 3] where tracking graphs took hours to compute and only for a single carefully selected threshold the accompanying movie demonstrates our ability to fully and interactively explore these datasets by varying thresholds, subselecting features and exploring

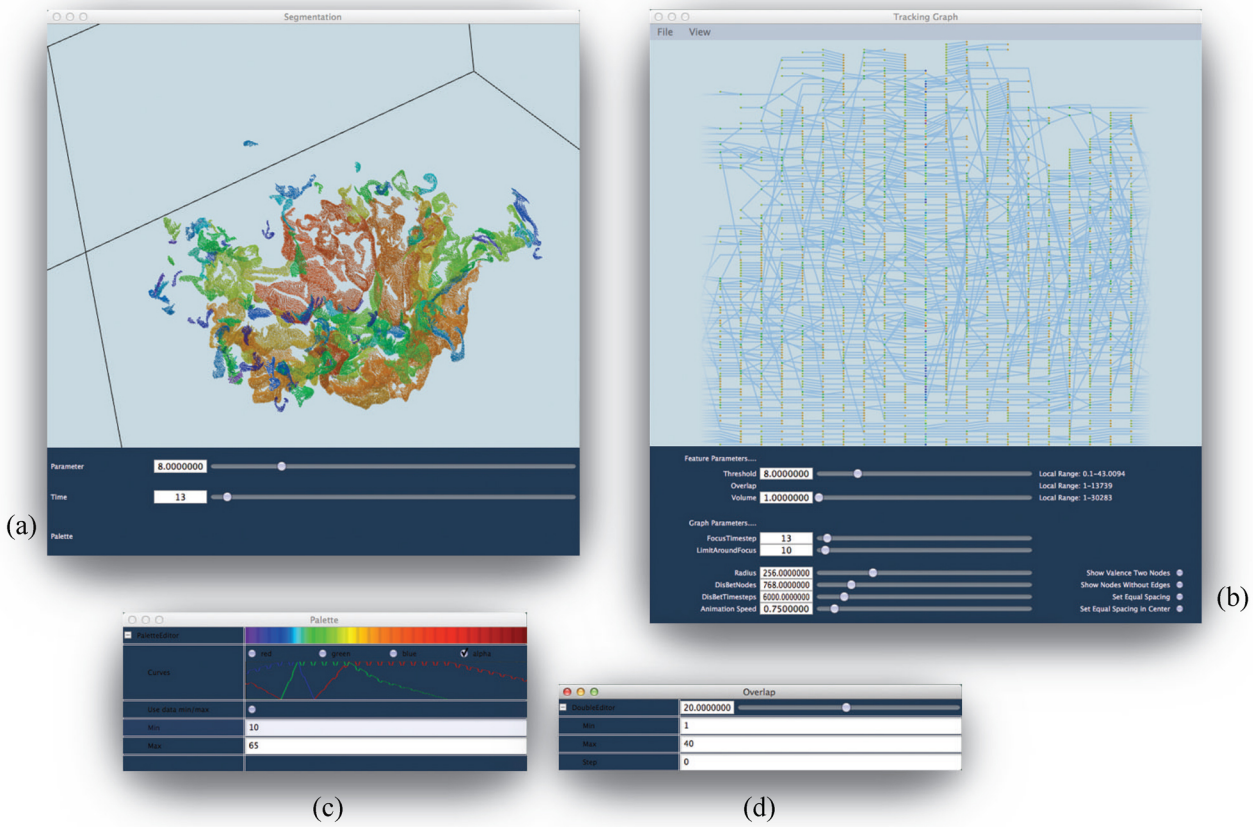

Figure 8: User interface of the system. The images displayed here use the Swirl dataset. (a) 3D segmentation viewer (b) Tracking graph viewer (c) Color palette editor used to change the current color scheme used (d) Slider editor for attributes.

various feature correlation thresholds.

The last dataset, temporalJet, describes a temporally evolving turbulent CO/H_2 jet flame undergoing extinction and reignition at different Reynolds numbers [22]. The simulations were performed with up to 0.5 billion grid points and periodic boundary conditions in the mean flow direction and we analyze 231 snapshots totaling roughly 1TB in raw data. The features of interest are local maxima of the scalar dissipation rate which form thin pancake-like regions under compressive turbulent strains. Unlike the previous two examples we do not use a threshold-based feature definition but rather a relevance threshold that locally adapts the threshold according to the higher maximum in the neighborhood. As shown in Figure 6(a) the resulting graphs are extremely dense and complex and virtually unmanageable with traditional graph layout tools.

Figure 5 shows a small portion of a graph in both the greedy as well as the optimized layout. Clearly the greedy layout is inferior, creating a large number of crossings, yet it still proves valuable for large graphs by providing users with immediate feedback about the their size and complexity. In addition, even in the greedy layout there exist noticeable subgroups of nodes which behave similarly and are laid out in parallel as a result of the natural temporal coherency present in the data. The greedy layout enables the user to quickly interrupt the rendering of needlessly complex graphs and helps guide the selection of better thresholds.

Figure 6 (a) & (b) shows a comparison between a layout computed in dot [26] and one produced by our system. Dot seems unable to reuse the space occupied by dying nodes and as a result creates a graph with extreme aspect ratios that is very difficult to explore let alone understand. Our system on the other hand manages to create a fairly compact layout with relatively few intersections. Overall, our layout does produce more intersections, yet the graph is still significantly more comprehensible.

8 Conclusion

We have presented a new framework to interactively explore features in massive time-dependent datasets. By creating a new meta-graph structure to encode families of graphs and using progressive and asynchornous graph layout algorithms we enable in-depth analyis of terabytes of data using commodity hardware. Nevertheless, there remain a number of open challenges in order to apply these techniques to the next generation of even larger simulations. First, the graph drawing remains restricted by the limited number of nodes and arcs a human can reasonably understand. Selecting subgraphs alleviates the problem but is contingent on finding features of interest beforehand. Therefore, new techniques are needed to find and highlight interesting features or events in massive graphs. Also if one can identify the critical timesteps in the entire dataset, visualization can even start with a compact view of the full tracking graph and then focus on one particular region of interest. This would be very useful when analyzing significantly large datasets. Furthermore, to complete the analysis capabilities of the tool we are planning to integrate more statistical views of the data along the lines of [2]. Finally, there exist more complex topological feature descriptors such as the Morse-Smale complex that may require

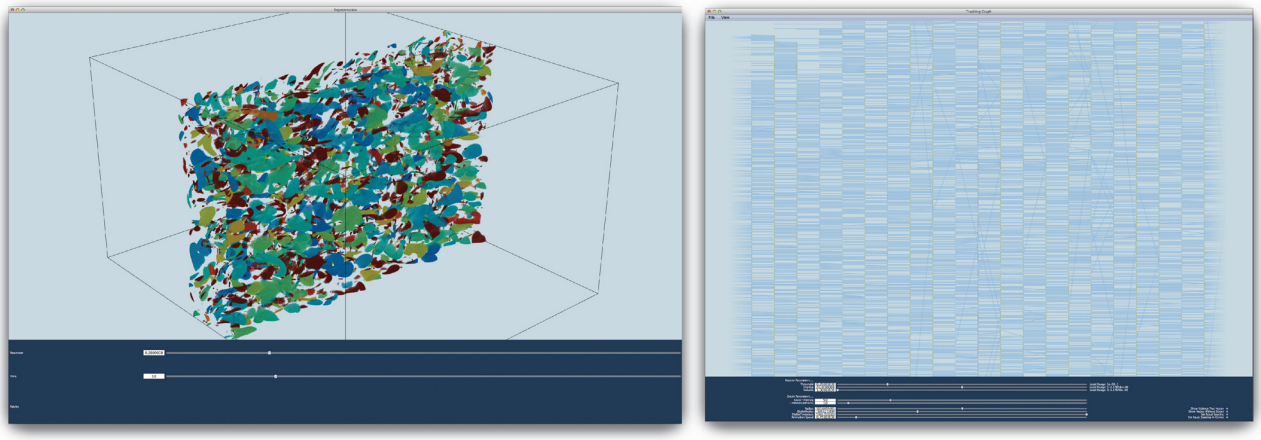

Figure 9: A optimal layout produced by the system for the TemporalJet dataset (focus timestep=50, threshold=0.2, overlap=20, volume=1.0)

different meta-graph structures.

Acknowledgements

This work is supported in part by NSF awards IIS-1045032, OCI-0904631, OCI-0906379 and CCF-0702817, and by a KAUST award KUS-C1-016-04. This work was also performed under the auspices of the U.S. Department of Energy by the University of Utah under contracts DE-SC0001922, DE-AC52-07NA27344 and DE-FC02-06ER25781, and by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 (LLNL-PROC-577473).

References

[1] C. L. Bajaj, V. Pascucci, and D. R. Schikore. The contour spectrum. In *Proceedings of the 8th conference on Visualization '97*, VIS '97, pages 167–ff., Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.

[2] J. C. Bennett, V. Krishnamoorthy, S. Liu, R. W. Grout, E. R. Hawkes, J. H. Chen, J. Shepherd, V. Pascucci, and P.-T. Bremer. Feature-based statistical analysis of combustion simulation data. *IEEE Transactions on Visualization and Computer Graphics*, 17:1822–1831, 2011.

[3] P.-T. Bremer, G. H. Weber, V. Pascucci, M. S. Day, and J. B. Bell. Analyzing and tracking burning structures in lean premixed hydrogen flames. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):248–260, 2010.

[4] P.-T. Bremer, G. H. Weber, J. Tierny, V. Pascucci, M. S. Day, and J. B. Bell. A topological framework for the interactive exploration of large scale turbulent combustion. In *Proceedings of the Fifth IEEE International Conference on e-Science '09*, E-SCIENCE '09, pages 247–254, Washington, DC, USA, 2009. IEEE Computer Society.

[5] C. Buchheim, A. Wiegele, and L. Zheng. Exact algorithms for the quadratic linear ordering problem. *INFORMS J. on Computing*, 22(1):168–177, Jan. 2010.

[6] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '00, pages 918–926, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[7] H. Carr, J. Snoeyink, and M. van de Panne. Simplifying flexible isosurfaces using local geometric measures. In *Proceedings of the conference on Visualization '04*, VIS '04, pages 497–504, Washington, DC, USA, 2004. IEEE Computer Society.

[8] M. Chimani, C. Gutwenger, P. Mutzel, and H.-M. Wong. Layer-free upward crossing minimization. *J. Exp. Algorithmics*, 15:2.2:2.1–2.2:2.27, Mar. 2010.

[9] M. Chimani, C. Gutwenger, P. Mutzel, and H.-M. Wong. Upward planarization layout. In D. Eppstein and E. Gansner, editors, *Graph Drawing*, volume 5849 of *Lecture Notes in Computer Science*, pages 94–106. Springer Berlin / Heidelberg, 2010.

[10] M. Chimani, P. Hungerländer, M. Jünger, and P. Mutzel. An sdp approach to multi-level crossing minimization, March 2011.

[11] M. Chimani, P. Mutzel, and I. Bomze. A new approach to exact crossing minimization. In *Proceedings of the 16th annual European symposium on Algorithms*, ESA '08, pages 284–296, Berlin, Heidelberg, 2008. Springer-Verlag.

[12] M. Day, J. Bell, P.-T. Bremer, V. Pascucci, V. Beckner, and M. Lijewski. Turbulence effects on cellular burning structures in lean premixed hydrogen flames. *Combustion and Flame*, 156:1035–1045, 2009.

[13] M. S. Day and J. B. Bell. Numerical simulation of laminar reacting flows with complex chemistry. *Combust. Theory Modelling*, 4(4):535–556, 2000.

[14] P. Eades and D. Kelly. Heuristics for reducing crossings in 2-layered networks. *Ars Combinatoria*, pages 89–98, 1986.

[15] P. Eades and N. C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11:379–403, 1994. 10.1007/BF01187020.

[16] H. Edelsbrunner, J. Harer, A. Mascarenhas, and V. Pascucci. Time-varying reeb graphs for continuous space-time data. In *Proceedings of the twentieth annual symposium on Computational geometry '04*, SCG '04, pages 366–372, New York, NY, USA, 2004. ACM.

[17] I. Fujishiro, Y. Maeda, and H. Sato. Interval volume: a solid fitting technique for volumetric data display and analysis. In *Proceedings of the conference on Visualization '95*, VIS '95, pages 151 –158, 448. IEEE Computer Society, oct-3 nov 1995.

[18] I. Fujishiro, R. Otsuka, S. Takahashi, and Y. Takeshima. T-map: a topological approach to visual exploration of time-varying volume data. In *Proceedings of the 6th international symposium on high-performance computing and 1st international conference on Advanced low power systems*, ISHPC'05/ALPS'06, pages 176–190, Berlin, Heidelberg, 2008. Springer-Verlag.

[19] T. Gerstner and R. Pajarola. Topology preserving and controlled topology simplifying multiresolution isosurface extraction. In *Proceedings of the conference on Visualization '00*, VIS '00, pages 259 –266. IEEE Computer Society, oct. 2000.

[20] S. Guthe and W. Straßer. Real-time decompression and visualization of animated volume data. In *Proceedings of the conference on Visualization '01*, VIS '01, pages 349–356, Washington, DC, USA, 2001. IEEE Computer Society.

[21] A. Gyulassy and V. Natarajan. Topology-based simplification for feature extraction from 3d scalar fields. In *Proceedings of the conference on Visualization '05*, VIS '05, pages 535 – 542. IEEE Computer Society, oct. 2005.

[22] E. Hawkes, R. Sankaran, J. Sutherland, and J. Chen. Scalar mixing in direct numerical simulations of temporally evolving plane jet flames with skeletal co/h2 kinetics. *In Proceedings of the Combustion Insti-*

tute, 31(1):1633 – 1640, 2007.

[23] A. Joshi and P. Rheingans. Illustration-inspired techniques for visualizing time-varying data. In *Proceedings of the conference on Visualization '05*, VIS '05, pages 679 – 686. IEEE Computer Society, oct. 2005.

[24] M. Jünger, E. K. Lee, P. Mutzel, and T. Odenthal. A polyhedral approach to the multi-layer crossing minimization problem. In *Proceedings of the 5th International Symposium on Graph Drawing*, GD '97, pages 13–24, London, UK, UK, 1997. Springer-Verlag.

[25] M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: performance of exact and heuristic algorithms. *J. Graph Algorithms Appl*, 1:1–25, 1997.

[26] E. Koutsofios and S. North. Drawing graphs with dot. Technical Report 910904-59113-08TM, AT&T Bell Laboratories, Murray Hill, NJ, 1991.

[27] Y. kun Zhang, H. Chen, D. xin Hua, Y. an Cui, and B. wei Zhang. An edge crossing minimization algorithm based on adjacency matrix transformation. In *Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering '10*, volume 1 of *ICACTE '10*, pages V1–672 –V1–675, aug. 2010.

[28] D. Laney, P. T. Bremer, A. Mascarenhas, P. Miller, and V. Pascucci. Understanding the structure of the turbulent mixing layer in hydrodynamic instabilities. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1053–1060, Sept. 2006.

[29] X. Y. Li and M. F. Stallmann. New bounds on the barycenter heuristic for bipartite graph drawing. *Inf. Process. Lett.*, 82(6):293–298, June 2002.

[30] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4):163–169, 1987.

[31] E. B. Lum, K. L. Ma, and J. Clyne. Texture hardware assisted rendering of time-varying volume data. In *Proceedings of the conference on Visualization '01*, VIS '01, pages 263–270, Washington, DC, USA, 2001. IEEE Computer Society.

[32] A. Mascarenhas, R. W. Grout, P.-T. Bremer, E. R. Hawkes, V. Pascucci, and J. H. Chen. Topological feature extraction for comparison of terascale combustion simulation data. In *Topological Methods in Data Analysis and Visualization*, Mathematics and Visualization, pages 229–240. Springer Berlin Heidelberg, 2011.

[33] J. Milnor. Morse theory. Princeton University Press, New Jersey, 1963.

[34] F. Reinders, F. H. Post, H. J. W. Spoelder, and K. V. Time-dependent. Visualization of time-dependent data using feature tracking and event detection. *The Visual Computer*, 17:55–71, 2001.

[35] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, ICCAD '93, pages 42–47, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[36] R. Samtaney, D. Silver, N. Zabusky, and J. Cao. Visualizing features and tracking their evolution. *Computer*, 27(7):20–27, July 1994.

[37] H.-W. Shen, L.-J. Chiang, and K.-L. Ma. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. In *Proceedings of the conference on Visualization '99*, VIS '99, pages 371–377, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

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

[39] D. Silver and X. Wang. Tracking scalar features in unstructured data sets. In *Proceedings of the conference on Visualization '98*, VIS '98, pages 79 –86. IEEE Computer Society, oct. 1998.

[40] B.-S. Sohn and C. Bajaj. Time-varying contour topology. *IEEE Transactions on Visualization and Computer Graphics*, 12(1):14 –25, jan.-feb. 2006.

[41] K. Stockinger, J. Shalf, K. Wu, and E. W. Bethel. Query-driven visualization of large data sets, 2005.

[42] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109 –125, feb. 1981.

[43] P. Sutton and C. Hansen. Isosurface extraction in time-varying fields using a temporal branch-on-need tree (t-bon). In *Proceedings of the conference on Visualization '99*, VIS '99, pages 147 –520. IEEE Computer Society, oct. 1999.

[44] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *Proceedings of the thirteenth annual symposium on Computational geometry '97*, SCG '97, pages 212–220, New York, NY, USA, 1997. ACM.

[45] Visus. http://www.visus.us/.

[46] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Trans. Graph.*, 11(3):201–227, July 1992.

[47] J. Woodring, C. Wang, and et al. High dimensional direct rendering of time-varying volumetric data. pages 417–424, 2003.