Title

Interactive Exploration and Analysis of Large-Scale Simulations Using Topology-Based Data Segmentation

Journal

IEEE Transactions on Visualization and Computer Graphics, 17(9)

ISSN

1077-2626

Authors

Bremer, P-T

Weber, G

Tierny, J

et al.

Publication Date

2011-09-01

DOI

10.1109/tvcg.2010.253

Peer reviewed

Appeared in IEEE Transactions on Visualization and Computer Graphics, vol. 17, no. 9, pp. 1307–1324.

Interactive Exploration and Analysis of Large Scale Simulations Using Topology-based Data Segmentation

Peer-Timo Bremer, Gunther H. Weber, Julien Tierny, Valerio Pascucci, Marcus S. Day, and John B. Bell

Abstract—Large-scale simulations are increasingly being used to study complex scientific and engineering phenomena. As a result, advanced visualization and data analysis are also becoming an integral part of the scientific process. Often, a key step in extracting insight from these large simulations involves the definition, extraction, and evaluation of features in the space and time coordinates of the solution. However, in many applications these features involve a range of parameters and decisions that will affect the quality and direction of the analysis. Examples include particular level sets of a specific scalar field, or local inequalities between derived quantities. A critical step in the analysis is to understand how these arbitrary parameters/decisions impact the statistical properties of the features, since such a characterization will help to evaluate the conclusions of the analysis as a whole.

We present a new topological framework that in a single pass extracts and encodes entire families of possible features definitions as well as their statistical properties. For each time step we construct a hierarchical merge tree a highly compact, yet flexible feature representation. While this data structure is more than two orders of magnitude smaller than the raw simulation data it allows us to extract a set of feature for any given parameter selection in a post-processing step. Furthermore, we augment the trees with additional attributes making it possible to gather a large number of useful global, local, as well as conditional statistic that would otherwise be extremely difficult to compile. We also use this representation to create tracking graphs that describe the temporal evolution of the features over time. Our system provides a linked-view interface to explore the time-evolution of the graph interactively alongside the segmentation, thus making it possible to perform extensive data analysis in a very efficient manner. We demonstrate our framework by extracting and analyzing burning cells from a large-scale turbulent combustion simulation. In particular, we show how the statistical analysis enabled by our techniques provides new insight into the combustion process.

1 Introduction

High resolution numerical simulations have become an integral part of the scientific process. They allow scientists to observe a range of phenomena not easily captured by experiments and are an essential tool to develop and validate new scientific theories. As the spatial and temporal resolution of these simulations increases, so does the need for efficient methods to visualize and analyze the resulting data. Traditionally, scientists use techniques such as isosurfaces [1, 2] to identify features of interest and their defining parameters. These features are then extracted and analyzed using a secondary tool chain. However, after each change in parameters, such as isovalue, the entire data must be re-processed, requiring significant time and computational effort. Even using acceleration look-up structures, such as the span-space [3], still requires access to the original simulation data for re-processing. Relying directly on the original data renders interactive techniques infeasible as common data sets easily reach several Terabytes in size.

To improve data analysis and visualization capabilities for high-resolution simulations, we propose a novel framework that uses pre-computed topological information to enable interactive exploration of large-scale data sets. We demonstrate the effectiveness of these techniques using simulations of a lean, pre-mixed hydrogen flame [4, 5]. These flames burn in a dynamic cellular mode that is characterized by intensely burning cells separated by regions where the flame is extinguished. Burning cells are defined via a threshold on the fuel consumption rate, and scientists are interested in the number, size, and temporal evolution of these cells. However, no single correct threshold is known, and studying the flame characteristics under different thresholds is one of the primary goals.

In prior work [6], we used the Morse complex defined on an isotherm (isosurface of temperature) to perform analysis over a wide parameter range of fuel consumption thresholds and provided the ability to track burning regions over time. However, since the analysis was performed on an extracted isotherm, it required specifying a temperature as an additional fixed parameter. In contrast, our new framework is based on: (i) The merge tree of fuel consumption rate, computed for the original three-dimensional data; (ii) The segmentation implied by the merge tree; and (iii) Additional derived statistics of the resulting features. By storing the merge trees hierarchically, we encode all possible cell segmentations in a single data structure, suitable for both three-dimensional and two-dimensional data. This data structure is two orders of magnitude smaller than the input, making it possible to explore interactively an entire family of features along with aggregate attributes, such as cell volume or average fuel consumption using pre-computed topological information. We also implement tracking of burning regions over time.

Splitting segmentation information from the hierarchical merge trees, we create a lightweight index into the pre-segmented data that can be loaded on demand, thus enabling interactive analysis. A linked-view system uses this index to correlate the tracking graph with displays of segmentations supporting interactive exploration of their temporal evolution. Using pre-computed attributes, such as cell volume or average fuel consumption, it is possible to sub-select cells and explore the 3D segmentation interactively. Based on these subsets, we aggregate pre-computed statistics for quantitative analysis. Thus, topological analysis allows one to fully explore the parameter space used for segmenting, selecting, and tracking burning cells as defined by the domain scientists. Furthermore, we demonstrate that the high level of abstraction of the topological representation, which reduces data by more than two orders of magnitude, does not impact adversely the functionality in the data exploration process. In particular, one can explore all possible segmentations of burning cells to understand better their dynamics as well as to validate the method. Finally, we demon-

- *P.-T. Bremer is with Center of Applied Scientific Computing (CASC), Lawrence Livermore National Laboratory, Box 808, L-560, Livermore, CA 4551 and the Scientific Computing and Imaging (SCI) Institute, University of Utah, 72 South Central Campus Drive, Salt Lake City, UT 84112. E-mail: ptbremer@acm.org.*
- *G.H. Weber is with the Visualization Group, Computational Research Division (CRD), Lawrence Berkeley National Laboratory, One Cyclotron Road, MS: 50F-1650, Berkeley, CA 94720 and the Institute for Data Analysis and Visualization, Department of Computer Science, University of California, Davis, One Shields Avenue, Davis, CA 95616. E-mail: GHWeber@lbl.gov.*
- *J. Tierny and V. Pascucci are with the Scientific Computing and Imaging (SCI) Institute, University of Utah, 72 South Central Campus Drive, Salt Lake City, UT 84112. E-mail: {jtierny,pascucci}@sci.utah.edu.*
- *M.S. Day and J.B. Bell are with the Center for Computational Science and Engineering (CCSE), Computational Research Division (CRD), Lawrence Berkeley National Laboratory, One Cyclotron Road, MS: 50A-1148, Berkeley, CA, 94720. E-mail {MSDay,JBBell}@lbl.gov.*

strate that the topological data representation is ideally suited for performing extensive data analysis by providing a compact, yet complete representation of features of interest.

We have tested and validated our techniques extensively by processing more than 8 Terabytes of raw data. In collaboration with application scientists we are actively using the system to form new hypotheses about the burning process. In particular, scientists have found that our technique can facilitate understanding the relationship between the size distribution of burning cells, the threshold of fuel consumption and turbulent intensity. We have found that streaming merge tree segmentation is highly flexible with respect to the input and works on any grid type. Since it needs no temporary data, it also supports on-the-fly simplification as well as culling (i.e., dropping points before processing).

In summary, our contributions to the state of the art of visualization and data analysis are:

- A novel streaming algorithm for computing merge trees related to the streaming Reeb graph computation algorithm introduced by Pascucci et al. [7]. Our new approach overcomes the unfavorable scaling in three dimensions, computes the corresponding segmentation on the fly, and works for any grid type.
- Computing tracking graphs correlating segmentations over time for user-specified thresholds.
- Augmenting the merge tree with pre-computed attributes such as average fuel consumption rate or burning cell volume.
- Utilizing merge trees for data compression enabling interactive analysis of ultra-scale simulations on desktop workstation or laptop computers. In particular, the merge trees serve as a very light weight index into the pre-segmented data that can be loaded interactively on demand for each parameter selection.
- A linked-view interface for exploring the relationship between data segmentation and time tracking graph for the features of interest. This system further utilizes pre-computed attributes such as cell volume to support high-level filtering operations on the segmentation.
- Aggregating pre-computed attributes into comprehensive statistics based on filtering criteria chosen in the linked-view system. This feedback is essential for validation of the system.
- Demonstrating the applicability of our system by performing a full 3D analysis on state of the art, large scale combustion simulations. This application shows that our topological analysis supports broad exploration of the parameter space used for segmenting, selecting, and tracking burning regions as defined by the domain scientists thus validating the effectiveness of our tool. In particular, users have found our technique to be critical in gaining new insight in the dynamics of the combustion process such as in the understanding the relationship of the size distribution of burning cells and the threshold of fuel consumption rate.

2 Related Work

Fundamentally, our work addresses the definition, analysis, and tracking of features. In flow and vector field visualization, feature extraction and tracking are still an open topic and the subject of ongoing research [8]. Our work, however, focuses on scalar field exploration, where a variety of basic feature definitions have proven valuable and successful. Isosurfaces [1, 2], interval volumes [9], or thresholding combinations of various scalar quantities [10, 11] often serve as the building blocks for defining features of interest. The resulting structures provide an abstract visualization tool and often have concrete physical interpretations. For example, in combustion simulations, isosurfaces of temperature are often equated with flame surfaces.

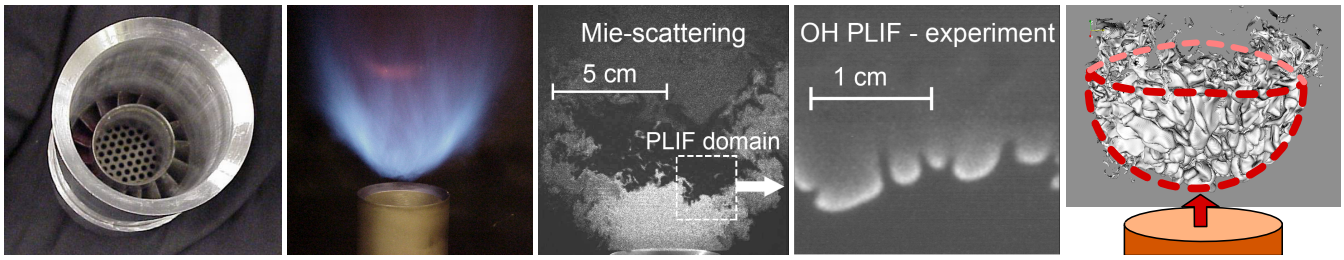
Defining and extracting features usually serves the purpose of deriving quantitative measurements, tracking those features of interest over time or both. For scalar data, most feature definitions are based on isosurfaces or thresholded regions. Mascarenhas and Snoeyink [12] provide a detailed overview of isosurface tracking. Essentially, tracking algorithms can be divided into two main categories: tracking by geometry and tracking by topology. Methods in the former category use distance between geometric attributes, e.g., the center of gravity [13] or volume overlap [14, 15], for tracking. Laney et al. [16] use a similar approach to track bubble structures mentioned in the feature detection section in turbulent mixing. Ji et al. [17, 18] track the evolution of isosurfaces in a time-dependent volume by extracting the 3D space-time isosurface in a 4D space. Finally, Weber et al. [19] creating tracking graphs for features embedded into time-dependent iso-surfaces by computing the Reeb graph of the space-time surface defined by feature boundaries.

Methods in the latter category [20, 21] compute tracking information topologically using, for example, Jacobi sets [22], which describe the paths all critical points take over time. Sohn and Bajaj [23] introduce a hybrid approach using volume matching similar to Silver and Wang [14, 15] instead of topological information [22, 21] to define correspondences between contour trees.

The application of such feature definitions to large scale data is hampered by the need for multiple static thresholds. A single set of features is extracted for each set of thresholds, and any change in a threshold requires re-processing of data. This approach makes visualization of features with variable thresholds difficult and the corresponding data analysis costly. Topological techniques [24] address this problem by expressing a similar set of features using concepts of Morse theory [25, 26]. Furthermore, Morse theory provides a well developed notion of simplification and hierarchical structures. This notion makes feature definitions hierarchical supporting noise removal and multi-scale analysis.

Features of interest may be defined using the Reeb graph [27, 7], the contour tree [28, 29, 30, 31, 32, 33] as well as the related volume skeleton tree [34, 35]) and the Morse-Smale complex [36, 37, 38, 39, 40]. For example, Carr et al. show the benefit of using individual connected components of an isosurface [41] and further show that contour tree simplification can pick up anatomical structures [42]. Weber et al. [43] use the same concept to segment medical and simulation data for volume rendering. Further examples of topological data analysis include using the Morse-Smale complex to detect bubbles in turbulent mixing [16] or the core structure of porous solids [44]. Takahashi et al. [45] introduce new distance metrics to manifold learning that support recasting contour tree calculation as a dimensionality reduction scheme. While this approach makes it possible to reduce data set sizes and segment data, it does not compute an explicit graph representation of the contour necessary for associating the tree structure with quantitative measurements.

Recently, we have used topological techniques to analyze extinction regions in non-premixed turbulent combustions [46] and to study lean pre-mixed hydrogen flames [6, 19]. Similar to [46] we use hierarchical merge trees to encode one-parameter families of segmentations. However, we extend the trees by associating additional attributes with the segmentation and use a different hierarchy construction as well. Their method also lacks the ability to visualize, let alone explore, resulting tracking graphs. Furthermore, unlike [46] we show how the resulting information can be used to represent compactly the features of interest for an entire simulation and demonstrate an interactive exploration framework based on this representation. In [6, 19] we analyze lean pre-mixed hydrogen flames considered to be an idealized, simpler version of the low-swirl flames that are the focus here. However, the analysis in [6] requires the use of an arbitrary isosurface, which can bias the results and makes feature tracking significantly more difficult [19]. Instead, we extend the analysis of [6] to three-dimensions removing the bias and add the ability to explore the entire simulation interactively. By performing a three-dimensional segmentation directly, feature tracking also becomes simpler, (see Section 5.3) eliminating the need for more advanced tracking techniques. Recently, we published preliminary findings focusing on the combustion application [47]. Here, we present an in-depth description of the novel algo-



(a) (b) (c) (d) (e)

Fig. 1. (a) Photo of a typical laboratory low-swirl nozzle. (b) Photo of a lean premixed CH_4 low-swirl flame. (c) Experimental Mie scattering image of a lean premixed H_2 flame. (d) PLIF data imaging the OH concentration in a lean premixed H_2 flame. (e) Rendering of the burning cells of the SwirlH2 simulation data. The cells form a bowl shaped structure with the arrow indicating the direction of the fuel stream.

rithms used in that study as well as new, further substantiated analysis results.

An important aspect of our system is that it correlates tracking graphs and segmented views via linking. Linking multiple views for the visualization of high-dimensional data sets is an established concept in information visualization [48]. For example, Henze [49] proposed a system for exploring time-varying computational fluid dynamics (CFD) data sets that uses multiple views (called Portraits in the paper) displaying a data set and various derived quantities. Users can perform advanced queries by selecting data subsets in these portraits. The concept of multiple views was also used in the WEAVE system, where a combination of physical views and information visualization views allows exploration of cardiac simulation and measurement data [50]. Doleisch et al. [51] generalize this concept with abstract feature definitions based on thresholds and linking physical views and information visualization views to facilitate data exploration. In the context of topological data analysis, Fujishiro et al. [52] introduced the T-Map that captures the topological structure of a 4D volume, shows topological changes in pixel map form, and supports effective drill-down operations. For example, their system supports highlighting regions identified in the T-Map in volume rendered images of the original time sequence data.

3 Application

Improving our ability to interpret diagnostics of complex, experimentally observable phenomena is an important application of modern large-scale numerical simulations. We explore this issue in the context of combustion, where detailed simulations are used to support the fundamental basis behind the interpretation of critical laser-based flame diagnostic approaches. For a detailed discussion on basic combustion theory we refer the reader to the book by Williams [53] and for the theory and numerical modeling of flames including turbulence to Poinsot and Veynante [54]. We focus this study on advanced ultra-lean premixed combustion systems, see Bell et al. [55] for a discussion of recent research in simulation of lean premixed combustion, and our ultimate goal is to: (i) Augment and validate laser-based diagnostics; (ii) Assess the underlying assumptions in their interpretation; and (iii) Aid the development of models to characterize the salient behavior of these flame systems.

Low-swirl injectors [4, 56, 5, 57, 58] are emerging as an important new combustion technology. In particular, such devices can support a lean hydrogen-air flame that has the potential to dramatically reduce pollutant emissions in transportation systems and turbines designed for stationary power generation. However, hydrogen flames are highly susceptible to various fluid-dynamical and combustion instabilities, making them difficult to design and optimize. Due to these instabilities, the flame tends to arrange itself naturally in localized cells of intense burning that are separated by regions of complete flame extinction.

Existing approaches to analyze the dynamics of flames, including most standard experimental diagnostic techniques, assume that the flame is a connected interface that separates the cold fuel from hot combustion products. In cellular hydrogen-air flames, many of the basic definitions break down—there is no connected interface between the fuel and products, and in fact there is no concrete notion of a "progress variable" that can be used to normalize the progress of the combustion reactions through the flame. As a consequence, development of models for cellular flames requires a new paradigm of flame analysis.

Fig. 1(a) shows the detail of a low-swirl nozzle. The annular vanes inside the nozzle throat generate a swirling component in the fuel stream. Above the nozzle the resulting flow-divergence provides a quasi-steady aerodynamic mechanism to anchor a turbulent flame. Fig. 1(b) illustrates such a flame for a lean premixed CH_4-air mixture (the illustration shows a methane flame since H_2 flames do not emit light in the visible spectrum). Figs. 1(c),1(d) show typical experimental data from laboratory low-swirl, lean H_2-air flames. Such data is used to extract the mean location and geometrical structure of instantaneous flame profiles. The images indicate highly wrinkled flame surfaces that respond in a complex way to turbulent structures and cellular patterns in the inlet flow-field.

In earlier studies, we presented a novel approach for a generalized analysis of cellular flames. The method was based on tracking time-dependent isotherms. A hierarchical segmentation strategy was used to carve the isotherms into connected flame pieces based on local rates of fuel consumption. These flame pieces were then tracked over the numerical evolution of the flow. For a given set of conditions, the size and intensity of the individual burning cells were computed and used to characterize directly the effects of turbulence. In the present study, we generalize the previous work to avoid the initial step of interpolating the quantity of actual interest (the rate of fuel consumption) to a convenient scalar isosurface. The cellular regions of intense burning are identified by thresholding the local consumption rate and we work directly with the resulting three-dimensional, time-dependent regions. The process may be regarded as a generalized subsetting strategy, whereby subregions of a computational result may be sampled, explored and categorized in terms of a volume of space with an arbitrary application-specific definition for its boundary. Since we are interested in regions of high fuel consumption we have identified merge trees which encode the topology of super-level sets, see Section 4, as appropriate data structure. Nevertheless, the same techniques would apply to split trees in cases where minima are of interest or with some extensions to contour trees which encode features of both high and low function values.

The computational model used to generate the simulation results explored in this study incorporates a detailed description of the chemical kinetics and molecular transport, thus enabling a detailed investigation of the interaction between the turbulent flow field and the combustion chemistry. The simulation was carried out using a parallel adaptive low Mach number combustion code LMC [59] with an INCITE grant for computational resources at the National Energy Research Scientific Computing (NERSC) Center. The combination of adaptive mesh refinement, a low Mach number model formulation and the computational resources available through INCITE enabled us to perform these simulations in a 25 cm^3 domain with an effective resolution of 2048^3. Results from the simulation are in the form of a sequence of snapshots in time of the state data. Each snapshot is arranged in a hierarchy of block-structured grid patches ordered by refinement level and tiling successively smaller regions of the domain with successively finer grid cells. The highest resolution is focused around the flame surface (or,

region of high fuel consumption) and regions of high vorticity, and the adaptive hierarchy of grid patches evolves with the time-dependent solution. As in the physical device, the low-swirl burner simulation achieves a statistically stationary flame in a time-dependent turbulent flow field above the inlet nozzle.

We consider two simulations, which we label SwirlH2 and SwirlH2Fast, respectively, having different flow profiles. The SwirlH2Fast case has a mean fueling rate of 2.5 times that of the SwirlH2 case. In the simulations, the time-dependent integrated inventory of fuel in the domain is used to monitor the developing flame. Auxiliary diagnostic quantities, such as local chemical production rate, thermodynamical properties, etc. are computed from the state, as necessary. Once a quasi-steady configuration is obtained, snapshots were collected at intervals of approximately 2ms and 1ms for SwirlH2 and SwirlH2Fast, respectively and used for the analysis here. The data sets consist of 332 and 284 snapshots for the slow and fast version, respectively, at an effective resolution of 1024^3. The resulting snapshots are roughly 12–20 Gigabytes in size totaling a combined 8.4 Terabytes of raw data. The main features of interest are the intensely burning cells defined by a threshold on the local fuel consumption rate. All regions with a local fuel consumption rate above this threshold are tagged as "burning." Note, however, that no single "correct" threshold exists, requiring that we characterize the influence of this threshold value on the resulting diagnostics.

4 Background

Many of the data structures and algorithms used in this paper are rooted in Morse theory [25, 26]. In particular, we use hierarchical merge trees as primary data structure and here we briefly review the necessary theoretical background.

4.1 Hierarchical Merge Trees

First, we recapitulate the definitions of hierarchical merge trees, their connection to previously used structures such as the Morse complex [6], and how they can be used to store one-parameter families of segmentations. The concepts and data structures used here are similar to those used previously for studying extinction regions [46]. However, as discussed below we use a different algorithm and hierarchy. Furthermore, we augment the merge trees with additional attributes.

Given a smooth simply connected manifold $\mathbb{M} \subset \mathbb{R}^n$ and a function $f : \mathbb{M} \to \mathbb{R}$ the *level set* $L(s)$ of f at isovalue s is defined as the collection of all points on $\mathbb{R}$ with function value equal to s: $L(s) = \{p \in \mathbb{M} | f(p) = s\}$. A connected component of a level set is called a *contour*. Similarly, we define *super-level sets* $LS(s) = \{p \in \mathbb{M} | f(p) \geq s\}$ as the collection of points with function value greater or equal to s and *super-contours* as their connected components. The merge tree of f represents the merging of super-contours as the isovalue s is swept top-to-bottom through the range of f, see Fig. 2(a). Each time the isovalue passes a maximum a new super-contour is created and a new leaf appears in the tree. As the function value is lowered super-contours merge represented in the tree as the joining of two branches. Departing slightly from standard Morse theory we will call a point p in a merge tree *regular* if it has valence two and *critical* otherwise. Note that, this definition of critical points ignores points at which contours split or change genus as well as local minima as these do not affect the structure of a merge tree. Consequently, there exist only three types of critical points: maxima with no higher neighbors, saddles with two or more higher neighbors, and the global minimum with no lower neighbors.

Each branch in the merge tree corresponds to a neighboring set of contours and therefore branches represent subsets of $\mathbb{M}$. Here, we are interested in regions of high fuel-consumption rate and use sub-trees above a given threshold to define burning cells. Given a threshold, t, for the fuel consumption rate, f, we determine the corresponding burning cells by (conceptually) cutting the merge tree of f at t creating a forest of trees. Each individual tree represents one connected burning cell, see Fig. 2(e). In practice, rather than cutting the merge tree and traversing sub-trees the same information is stored more efficiently as a simplification sequence. A merge tree is simplified by successively merging leaf branches with their sibling branch. We order these simplifications by decreasing function value of the merge saddles and store the resulting simplification sequence. In this framework burning cells at threshold t are defined as sets of all leaf branches with function value greater than or equal to t of the tree simplified to t, see Figure 2(f).

Alternatively, this simplification can be viewed as the function value based simplification of maxima in the corresponding Morse complex [6]. In this context, the leaf branches above threshold t correspond to the stable manifolds of the Morse complex above t after canceling all saddles with function value $\geq t$ with neighboring maxima. However, the Morse complex used previously [6], by definition encodes a gradient based segmentation rather than a threshold-based one. Thus, the Morse complex needs a secondary data structure within each Morse cell to encode the segmentation while the merge tree naturally provides this information. Furthermore, as discussed below, the merge tree can be computed significantly more efficiently than the Morse complex making it the data structure of choice for three-dimensional data.

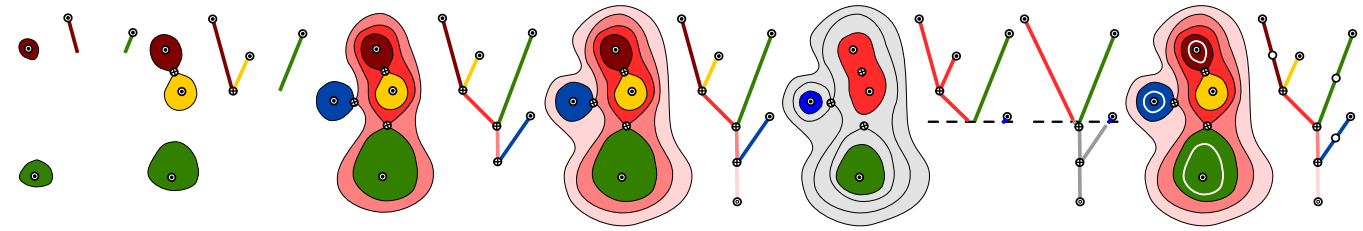
4.2 Augmented Merge Trees

Storing only the merge tree, the scheme described above allows us to determine the number of burning cells for all possible thresholds. However, in practice we also need an accurate representation of cell geometry and of any number of additional attributes such as volume (see below). Using only the original segmentation this is difficult since we must exclude the lower portions of branches intersecting the threshold. As the distribution of branch attributes can, in general, not be predicted, excluding a portion of a branch would require us to access the original data at significant cost. Instead, we extend the concepts introduced in [46] to augment the merge tree with additional valence two nodes by splitting all branches longer than some threshold, as seen in Fig. 2(g). Furthermore, unlike [46] we also compute a number of additional attributes for each branch. For example, we compute the volume of each branch as well as a number of k-th order moments such as means and variances for any variable of interest (not necessarily just f), see Section 5. As discussed below, this splitting is performed with little overhead during the initial computation and allows us to approximate cutting at any threshold with a pre-defined accuracy. It is important to note that this approximation only affects the geometry of the segments and their attributes but not their structure. We are guaranteed to not erroneously merge or split cells due to the approximation. In particular, augmenting the merge tree in this manner is a significant improvement over the regular sampling grid used in [6] that stores area values for all Morse complex cells for the entire function range independent of the local function range within that cell.

The augmented merge trees form the fundamental data structure in our framework, storing the one-parameter family of possible segmentations along with an arbitrary number of attributes. For efficient access during the visualization we store the segmentation information, a list of vertices per cell, separately. Even for the largest data sets the resulting merge trees consist of only around 6Mb ASCII information per time step compared to several Gigabytes of raw data. In fact, the trees are small enough to be loaded interactively from disk, see Section 6.

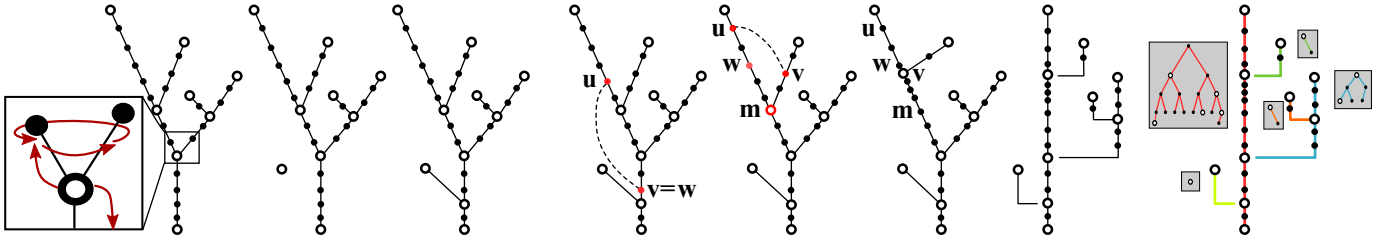
5 Data Processing

A key advantage of our system is the ability to represent more than a single static segmentation. Instead, we allow the user to browse the one-parameter family of all possible segmentations interactively. Furthermore, we provide various conditional attributes alongside the primary segmentation. This flexibility is based on computing hierarchical merge trees representing all possible segmentations and augmenting them with conditional information. The resulting data structure forms a highly flexible *meta-segmentation* that we use for visualization as well as data analysis. This section discusses the different algorithms and data structures used for processing individual time steps as well as constructing global tracking graphs.



(a) (b) (c) (d) (e) (f) (g)

Fig. 2. (a)-(d) Constructing a merge tree and corresponding segmentation by recording the merging of contours as the function value is swept top-to-bottom through the function range. (e) The segmentation for a particular threshold can be constructed by cutting the merge tree at the threshold, ignoring all pieces below the threshold and treating each remaining (sub-)tree as a cell. (f) The segmentation of (e) constructed by simplifying all saddles above the threshold. (g) The merge tree of (d) augmented by splitting all arcs spanning more than a given range.



(a) (b) (c) (d) (e) (f) (g) (h)

Fig. 3. Streaming merge tree construction: (a) Intermediate merge tree with the zoom-in showing the pointers used to maintain the data structure in red. Each vertex stores a pointer to its sibling, creating a linked list, as well as a pointer to one of its parents and a pointer to its child; (b) Merge tree of (a) after the addition of a new vertex; (c) The tree of (b) after attaching the lonely vertex with an edge; (d) Adding the dotted edge from u to v does not change the merge tree as v is already a descendant of u; (e) Adding the dotted edge from u to v creates a (temporary) loop which is immediately closed by merging the sub-branches $w-m$ and $v-m$. This creates a new saddle at v and changes m to a regular vertex shown in (f); (g) A branch decomposition of the tree in (f). (h) The branch decomposition of (g) where each branch stores a balanced search tree of its vertices.

5.1 Streaming Merge Tree Construction

We compute the augmented merge trees in a streaming fashion. The algorithm is similar in spirit to the one presented in [7] adapted to merge trees and improved to avoid the poor scaling for volumetric data. In general, streaming algorithms are attractive for large scale data processing due to their low memory foot print, high performance, and their ability to avoid temporary files and also to reduce file I/O. The streaming merge tree algorithm presented here provides several additional advantages. By representing the input data as a stream of vertices and edges it naturally supports culling of vertices outside a given function range of interest (in this case very low function values for example). Furthermore, we can perform an on-the-fly simplification that significantly reduces the size of the merge trees. Finally, a streaming approach is more flexible with respect to input formats and interpolation schemes. For example, as discussed in Section 7, processing data in its native adaptive mesh refinement (AMR) format provides a significant performance increase. We assume our input consists of a stream of vertices, edges between vertices, and finalization flags, where a vertex v must appear before the first edge that references v and the finalization tag of v appears after the last edge that uses v.

Algorithm. In the following discussion we use the nomenclature of vertices and links for the dynamically changing merge tree and nodes and arcs for elements of the final tree, which for distinction we will call merge graph. Note that vertices of the dynamic merge tree have a natural one-to-one correspondence to vertices of the input stream. The algorithm is based on three atomic operations corresponding to the three input elements: *CreateVertex* is called each time a vertex appears in the input stream; *AddEdge* is called for each edge in the input; and *FinalizeVertex* is called for each finalization flag. We illustrate the algorithm using the examples shown in Fig. 3. At all times we maintain a merge tree consisting of all (unfinalized) vertices seen so far. Each vertex stores its links using three pointers: A *down* pointer to its (unique) lower vertex; An *up* pointer to one of its higher vertices; and A *next* pointer to one of its siblings, see Fig. 3(a). The up and next pointers form a linked list of higher vertices allowing to completely traverse the tree. *CreateVertex* creates a new isolated vertex, see Fig. 3(b). *AddEdge* connects two vertices (u,v) of the existing tree and wlg. we assume $f(u) > f(v)$. With respect to the merge tree structure, each edge (u,v) provides one key piece of information about f: The existence of (u,v) guarantees that u's contour at $f(u)$ must evolve (potentially through merging) into v's contour at $f(v)$ and thus v must be a descendant of u in the tree. In the following we check this *descendant* property and if necessary adapt the merge tree to ensure its validity. First, we find the lowest descendant w of u such that $f(w) \geq f(v)$. If $w = v$ then the current tree already fulfills the descendant property and it remains unchanged , see Figs. 3(c) and 3(d). However, if $w \neq v$ the current tree does not fulfill the descendant property and must be modified. Another indication of this violation is that adding the new edge to the tree would create a loop which cannot exist in a merge tree, see Fig. 3(e). To restore a correct merge tree that reflects the new descendant information we perform a merge sort of both branches starting at w and v respectively until we find the first common descendant m of u and v, see Fig. 3(f). In this merge sort step, all other branches originating from a saddle are preserved. The merge sort closes the (illegal) loop and creates the correct merge tree. The pseudo code of the *AddEdge* algorithm is shown in Fig. 4

```
ADDEDGE(Vertex u, Vertex v, MergeTree T, Function f)
    if f(u) < f(v)
        SWAP(u, v)
    endif
    w = u
    while f(T.GETCHILD(w)) ≥ f(v) // Find the lowest child w
        w = T.GETCHILD(w)         // of u such that f(w) ≥ f(v)
    endwhile
    if w ≠ v   // If v is not a direct descendant of u
        T.MERGESORT(w, v) // Close the loop w, v, m (see Fig, 3(e))
    endif
```

Fig. 4. Pseudo-code for the *AddEdge* function to update a merge tree T to include the edge (u,v)

As vertices become finalized we remove them from the tree. However, care must be taken that this early removal does not impact the outcome of the computation. For example, a finalized saddle may become regular through global changes in the tree, see below. In partic-

ular, critical points of the merge tree must be preserved as they may form the nodes of the merge graph. In this context it is important to understand how the local neighborhood of a vertex relates to it being critical (wrt. to the merge tree, see Section 4). Here, we are interested in the connected components of the *upper link* [36] of a vertex v. Here the upper-link is defined as all vertices v_i connected to v with $f(v_i) > f(v)$ and the edges between them. By definition, a saddle must have more than one component in its upper link, while the upper link of the global minimum is a topological sphere of dimension $n-1$. However, the reverse is not true: A vertex can have multiple components in its upper link yet not be a (merge-tree) saddle (its upper link component may belong to contours that merged earlier); Similarly, the upper link of a local minimum is a topological sphere yet it is not critical with respect to the merge tree. This relationship becomes important when deciding which finalized vertices can be removed safely.

Finalizing a vertex v indicates that its entire local neighborhood has been processed. Thus, a finalized regular vertex will remain regular as more edges are added and can be removed without impacting the tree. Similarly, a finalized maximum will remain a leaf of the tree and thus will always correspond to a node of the merge graph. A finalized saddle, however, may become regular as its branches can be merged through additional edges not incident to the vertex itself. Finally, the global minimum of all vertices seen so far may become regular or a saddle when further vertices or edges are added. Nevertheless, a branch of the merge tree which consists only of a finalized leaf and a finalized saddle/minimum is guaranteed to be an arc of the merge graph (none of the contours it represents can be changed anymore). These conditions result in the algorithm shown in Fig. 5. When finalizing a vertex v we first check whether it is a regular vertex and if so remove it from the tree. We then determine whether this finalization has created a leaf edge between two finalized critical vertices. All such edges must be arcs in the merge graph and are added alongside their nodes. This procedure peels finalized branches from top to bottom from the merge tree and adds them to the merge graph. Finally, we check whether only the global minimum is left and remove it if necessary.

```
FinalizeVertex(Vertex v, MergeTree T, MergeGraph Final)
    T.MarkAsFinalized(v)
    c = T.GetChild(v)
    if T.IsRegular(v)
        T.BypassVertex(v) // Link all parents of v to its child c
        T.RemoveVertex(v)
    endif
    if T.IsCritical(c) AND T.IsFinalized(c)
        forall p ∈ T.GetParents(c)
            if T.IsCritical(p) AND T.IsFinalized(p)
                    AND T.GetParents(p) == ∅
                Final.AddArc(c, p)
                T.RemoveEdge(c, p)
                T.RemoveVertex(p)
                endif
            endif
        endfor
        if T.GetChild(c) == ∅ AND T.GetParents(c) == ∅
            T.RemoveVertex(c)
        endif
    endif
```

Fig. 5. Pseudo-code for the *FinalizeVertex* function to update a merge tree T after finalizing vertex v.

Optimization As described above, the algorithm can be seen as an optimized version of the one in [7]. By specializing the basic concept for merge trees one no longer needs to keep track of the interior of triangles (they do not influence the merging of contours) nor does one need to maintain loops in the graph as, by definition, merge trees cannot contain loops (independent of the genus of their input domain). However, applying this basic scheme to large scale three-dimensional data reveals a fundamental flaw in the original algorithm.

For each edge (u,v) we must find u's lowest descendant above v we call w. This requires us to traverse all vertices part of u's contour as we sweep through the range $[f(u), f(w)]$. Implemented in a straightforward fashion as shown in Fig. 4, this traversal takes time linear in the size of the contour and thus resulting in a worst-case time complexity of $O(\#edges * \#vertices)$. For triangulated surfaces with n vertices the expected number of edges is $3n$ and the expected size of a level set is $O(n^{\frac{1}{2}})$ [30] making the streaming Reeb graph algorithm scale virtually linearly for surfaces. For volumes, however, the expected size of a level set increases to $O(n^{\frac{2}{3}})$ which significantly impacts the scaling observed in practice. Furthermore, when combined with an expected $12n$ edges and the larger overall number of vertices the observed performance of the original algorithm decreases dramatically for higher dimensions.

To address this issue, we propose to rearrange the dynamic merge trees into a branch decomposition [60] and to maintain a balanced search tree for each branch, see Fig. 3(g). Clearly, in the new data structure the search for w takes time logarithmic in the size of the contour resulting in an expected search complexity of $O(n\log(n^{\frac{d-1}{d}}))$ for dimension d. To maintain the search structure, the time complexity to add and finalize a vertex increases to $\log(b)$ where b is the size of the branch. Furthermore, the merge sort increases in complexity from $O(b)$ to $O(b\log b)$ since vertices must be removed and added to the search structure. Thus, a conservative upper bound on the worst-case time complexity of the new algorithm is $O(n^2\log n)$. However, in practice, we find that the search for w is by far the most common (and expensive) operation and merge sorts typically handle very short segments of branches making the improved algorithm orders of magnitude faster than the naive implementation described above.

The new algorithm maintains all the flexibility of the streaming Reeb graph algorithm without the associated performance penalties. In particular, unlike the algorithm used in [46] we do not rely on regular grids as inputs nor do we need to pre-sort the data. As discussed in Section 7, extracting a regular subset of data from the given AMR format is far more expensive than the merge tree computation itself and its cost as well as its temporary storage can be avoided entirely using the streaming approach.

Vertex Connectivity Another important aspect when computing merge trees is the choice of triangulation or more generally the choice of neighborhood for each vertex. Traditionally, a volumetric grid would be tetrahedralized using the standard one-to-six voxel-tetrahedra split. However, this approach implies node-centered data and linear interpolation between nodes. In our specific application the data is cell centered and piece-wise constant. While it is easy to treat cells implicitly as nodes, imposing linear interpolation distorts the data. In particular, using a tetrahedralization introduces a grid bias where certain cells connected via their corners are considered neighbors (and have an edge between them) while others are not. We resolve this inconsistency by using the full 26-neighborhood around a cell when creating edges. While this connectivity no longer corresponds to any particular interpolation scheme (which it should not as the data is simulated to be piece-wise constant) it preserves the intuitive condition that two neighboring cells cannot both be local maxima. For a more rigorous argument of why this choice is appropriate we refer the reader to Carr and Snoeyink [61] who among other things show that choosing the 26-neighborhood is equivalent to a particular choice of marching cubes case tables. Nevertheless, as all our algorithms take an arbitrary set of edges as input we can switch easily to any other definition of neighborhood, e.g., tetrahedral or 6/12-neighborhood, depending on the application.

5.2 Segmentation

We compute the segmentation in a two stage approach: The first pass stores a pre-segmentation, which is completed in a second pass. During the initial merge tree computation we create a new segment each time a critical point is created and identify the segment with the mesh index of the corresponding vertex. As we traverse the tree during merge operations we pass on these labels from parent to child avoiding

virtually any computational overhead. As a result, each vertex knows its current *representative* (its next highest critical points), see Fig. 6(a). Maintaining the representatives is akin to a union-find except that sets can split. Since the notion of a representative is global (potentially the representative for all vertices can change at once) yet labels are updated only during merge sorts some labels may not be correct at any given time, see Fig. 6. In particular, the representative of a vertex v can change in two ways: v's initial representative u may become a regular vertex with representative w thus ceasing to be considered a representative, see Fig. 6(b). This corresponds to a union of the vertices with representative u with those of representative w. Conversely, a new saddle s may been created below u, see Fig. 6(c), which corresponds to splitting the set of vertices with representative u into two pieces. In the first case, u must have been part of the merge and thus stores the label of its correct representative w which has also become v's representative. In the second case, v's correct representative s can be found as the last saddle below u and above v. In general, there can exist any number of combinations of these two cases (see Fig. 6(b-d)) that cause labels to be out-dated. In particular, labels can be incorrect at the time a vertex is finalized and removed from memory. In this case we simply store its current label which we will correct in a second pass. It is important to point out that the representative of v can change after v has been finalized (and written to disk). Thus even correcting the labels during finalization would not protect against incorrect labels being recorded initially. Nevertheless, in practice labels are updated quite regularly and thus the pre-segmentation represents a good initial guess for the correct segmentation.

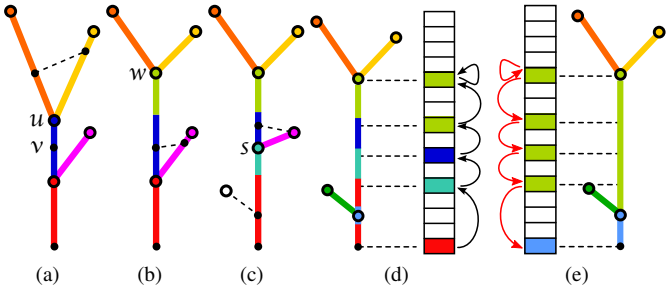


Fig. 6. Two-pass segmentation corresponding to a merge tree with critical points colored according to the branches they represent: (a) An initial tree segmented into branches indicated by color; (b) Adding the dotted edge shown in (a) results in a partial merge of branches causing the blue saddle u to become regular and a new green saddle w to appear; (c),(d) Adding further edges creates and destroys additional critical points as well as the corresponding segments. Note that labels are updated only as a side-effect of necessary merge-sorts and thus incur virtually no additional computational cost; (d) Using the indices of critical points as labels, the labels form linked lists of vertices terminating in a critical point above, representing the head of a linked list; (e) Traversing the implicit linked list in reverse order while traversing the merge graph (containing only critical points) allows one to correct all labels along the path.

We correct the pre-segmentation in a second pass by adjusting first for former critical points that are now regular and then for additional saddles. Since the representative label corresponds to a mesh index, one can use the stored labels as pointers into the array of vertices, see Fig. 6(d). For each vertex, we follow the chain of label pointers upward until we find a critical point (indicated by a vertex pointing to itself). This step corrects for any regular vertices that used to be critical. We then process the chain in the opposite direction while simultaneously traversing the nodes and arcs of the merge tree, see Fig. 6(e). In this manner we can determine the correct labels for all vertices in the chain. Notice that, after a vertex as been touched once, it stores a pointer to a critical point shortcutting any chain it might be part of in the future.

Nevertheless, this second pass re-traverses all vertices and is currently not performed in a streaming manner. While the merge tree is, by construction, stored in a streaming fashion the chains of vertices created during the traversal can represent a random access into the array of vertices. To avoid loading the entire array of vertices into memory and thus losing the advantage of a small memory footprint, we store the pre-segmentation as a memory mapped file. Consequently, the native paging system will ensure that any piece of the array necessary for the traversal is available. This strategy supports processing (almost) arbitrarily large data sets independently of the available main memory but potentially causing extensive disk swapping. In practice, the segmentation contains significant natural coherency and the pre-segmentation is accurate enough for this strategy to perform well. Finally, it is important to point out that when vertices are eliminated from the input stream before the computation they do not need to be stored and thus the segmentation will only be computed on the relevant subset of vertices.

The second stage of the segmentation process also provides a convenient way to adjust the segmentation to any simplification and/or splitting. After computing the initial merge tree, we can simplify the tree as discussed above. In that case, we reassign the segmentation labels of the simplified critical points to the critical point of their sibling arc. Furthermore, augmenting the tree with valence-two nodes is straightforward since the new segments are naturally introduced during the downward traversal. Finally, we use the second stage traversal to compute various attributes of the segments such as volume or average function value. In particular, we use the numerically stable parallel statistics algorithms of Bennett et al. [62] to compute various kth-order moments of the data. Note, that these computations are not limited to the function used to define the merge tree. Instead, we can pass an arbitrary number of secondary properties, e.g., temperature, mass fraction of various species etc., through the segmentation. As a result, we obtain these statistical properties conditioned on the primary segmentation allowing us to later compute a large number of conditional statistics, see Section 6. The resulting values are added to the hierarchical merge trees. We store the attribute itself, e.g., average fuel consumption, as well as all necessary information to correctly combine the attributes of two segments, e.g., current average of fuel consumption plus number of cells in the segment. This allows the interface to update accurately and interactively the attributes as the user changes the fuel consumption threshold. Finally, the segmentation is stored as a flat binary file of one label per vertex. As before, only vertices not culled from the stream are stored.

5.3 Feature Tracking

Given the hierarchical merge trees and corresponding segmentations for all time steps we track features defined by a given static threshold. Note that in this context *all* time steps should be understood as *all available* time steps. Due to the cost of I/O operations and limited disk-space, simulations typically save only a small fraction of the actual time steps. In our particular application past experience [19] has shown that the available temporal resolution is on the lower end of what is required for adequate tracking and thus we use all given data. We track features by spatial overlap which appears to be adequate for our purposes. However, since we have a complete description of features the framework can be easily extended to any number of more sophisticated techniques. For example, one may track the direction of the center of gravity [13] of a feature for a prediction / correction type of tracking or use schemes that rely on time interpolation [19]. To determine the potential overlap we load the merge trees of two consecutive time-steps and adapt them to the given threshold. We then traverse the vertices of both segmentations in parallel determining their active segmentation index and if both vertices are above the threshold add a (tracking graph) arc between the corresponding features. The active segmentation index is computed as the segmentation index stored in the file adapted to threshold simplification. Since the simplification consists of a sequence of merge operations adapting a segmentation index corresponds to a union-find style search for the highest un-simplified ancestor. Note that we could adjust easily the threshold to be time-dependent as each merge tree is handled separately.

Due to the large number of timesteps involved creating a tracking graph cannot yet be performed interactively. Furthermore, creating a layout for a given graph, even using state of the art tools such as

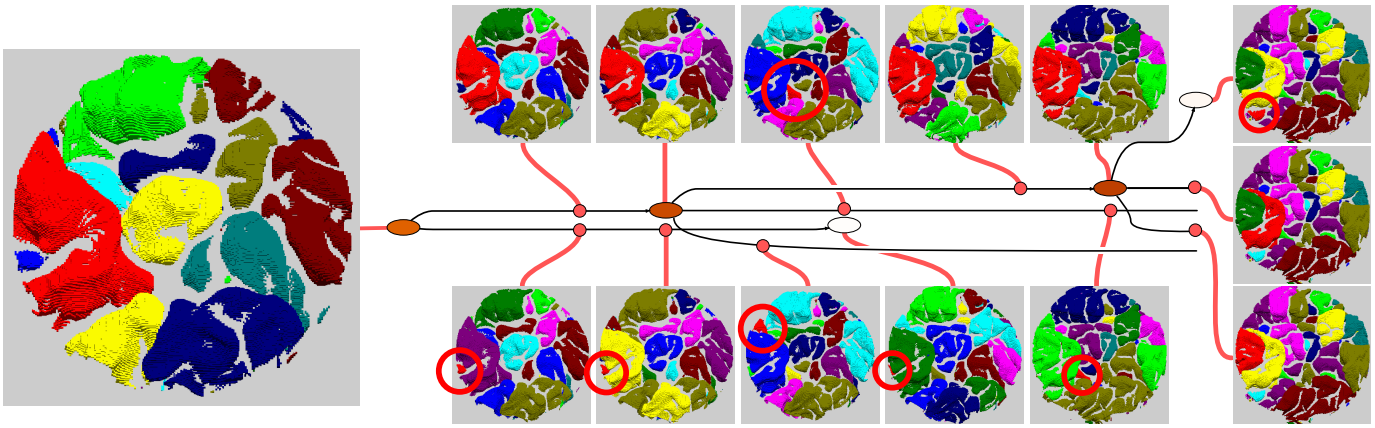


Fig. 7. Example of burning cells being tracked over time. The graph shows a small portion of the tracking graph for the SwirlH2 data set spanning time steps 1420 through 1465. The embedded screen shots show the corresponding segmentation as different nodes are selected in the graph. Over the course of these time steps the large cell represented by the left most node slow sheds smaller cells until it finally breaks apart into three independent pieces.

dot [63], remains too slow for interactive techniques. However, it is important to point out that the tracking graphs are created from the pre-computed segmentations and not the original data so all processing involved can easily be handled by a standard desktop computer.

For static grids the tracking is by design a streaming process. Each vertex is treated independently and since the vertices are in the same order for both time steps only two consecutive merge trees must be kept in memory. For each time interval we dump the partial tracking graph to disk to be assembled at the end. For time-dependent meshes such as AMR grids this strategy may fail as grids and thus vertex order in two consecutive time steps may differ. In this case we also store an index map for each vertex that maps its local index to a (virtual) global grid. During the overlap computation the vertices of both time steps must be mapped into global index space before an accurate tracking is possible. In this case we store one index map as a hash table in memory while streaming through the other index map along side the segmentation. The index maps are naturally in the same order as the segmentation files and are stored analogously as a flat binary file containing one index per vertex.

An example of features getting tracked through time is shown in Figure 7. The figure shows a small portion of the tracking graph for the SwirlH2 data set for time steps 1880 through 1895. The embedded screen shots show the main segmentation display when the indicated node has been selected. For a live screen capture of such a selection being done interactively we refer the reader to the accompanying movie.

5.4 Graph Simplification

As can be seen in the accompanying material, the tracking graphs can become highly complex and difficult to understand. Furthermore, they contain artifacts of the thresholding such as tiny features existing for only one or very few time steps. To reduce the graph complexity and eliminate some of the artifacts we simplify the tracking graphs by removing all valence zero nodes as well as nodes with a volume smaller than a given threshold. Since the original data is piece-wise constant we use the number of vertices corresponding to each node as a measure of volume. During the interactive session recorded in the movie one can clearly see how removing small features substantially unclutters the segmentation leaving only the larger features of interest. Similarly, such simplification significantly streamlines the tracking graph by suppressing unnecessary details. In order to avoid disconnecting segments above the volume threshold and thus structurally changing the tracking graph we restrict the simplification to successively removing leafs below the volume threshold.

To choose an adequate volume threshold we study how the number of nodes in the tracking graph changes as we increase the volume threshold, see Fig. 8. The graphs show a clear separation between noise and features. To reduce the complexity of the graphs and the cost of the layout we chose values on the upper range of the suggested noise level for simplification. All examples shown here use a threshold at around 100 vertices.

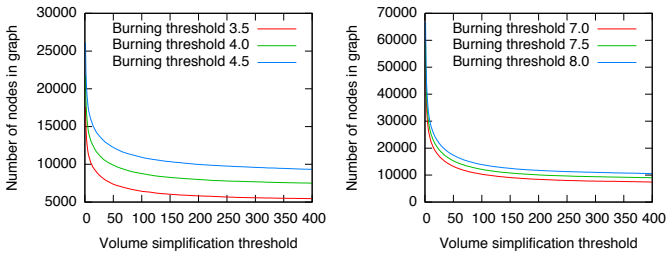


Fig. 8. Determining the tracking graph simplification threshold: The graphs show the number of nodes remaining in the tracking graphs vs. the simplification threshold for the restricted portions of the SwirH2 (top) and SwirlH2Fast (bottom) data set.

6 Interactive Exploration

The primary focus of our work is to provide the application scientists with the ability to comfortably explore their data in a manner meaningful in their particular problem space. Visualization is an important tool to validate simulations as well as to deliver a high level understanding of the underlying phenomena. Furthermore, it is a powerful method to investigate and setup further in-depth data analysis. For example, allowing the user to explore easily variables conditioned on the extracted features provides a simple way to understand whether various conditional statistics may provide new insights into the data.

To understand turbulent combustion, traditional visualization techniques are only of limited help. For example, while iso-surfaces at various thresholds would deliver geometry similar to our segmentations they contain none of the key information about whether two burning cells are connected, how many individual components exists, and/or their sizes, average values etc.. Clearly all this additional information could be computed but it would require accessing the original data at 16 Gigabytes per time step which would make any interactivity impractical. The hierarchical merge trees instead pre-process the data with respect to one of the most important aspects of the data (the burning cells) and store all additional information in accordance with this segmentation. Even for the more complicated SwirlH2Fast data set the resulting information only requires a roughly 6Mb ASCII file describing the hierarchical merge trees including attributes, a 144Mb segmentation, and a 144Mb index-map file per time step. Using standard gzip compression, these reduce to roughly 70Mb, which corresponds to a data reduction of more than two orders of magnitude, while still providing greater flexibility in the segmentation and selection than possible using standard techniques.

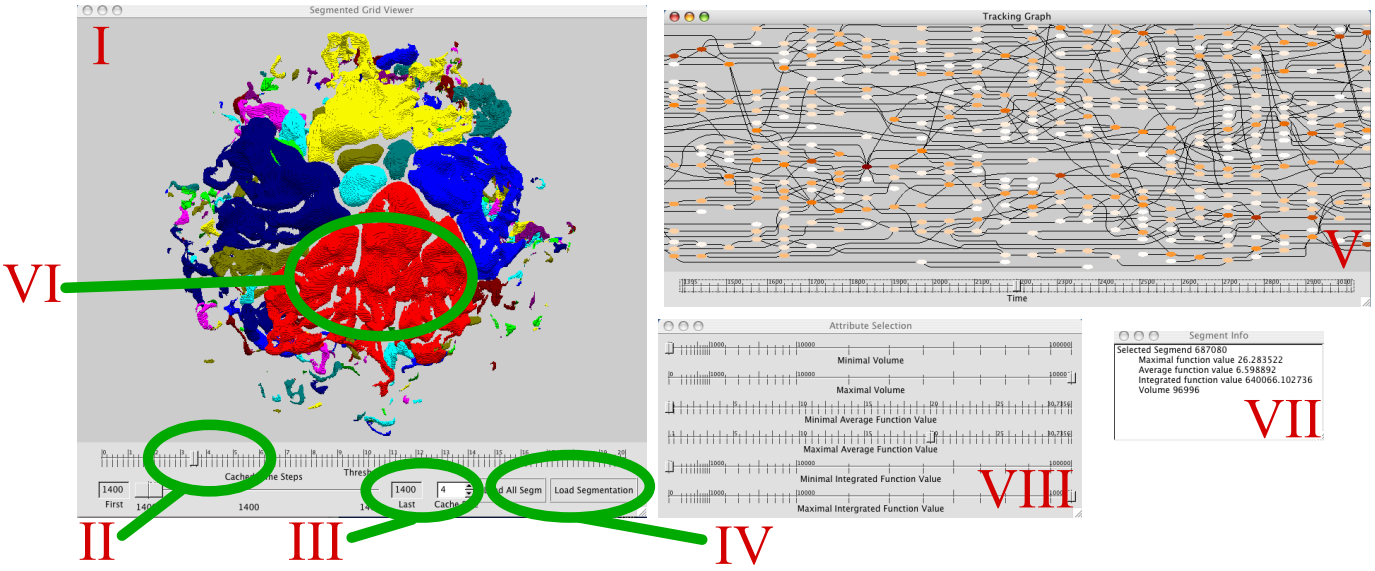


Fig. 9. Illustration of the different components of the user interface. (I) 3D display of the segmentation including a slider to select the fuel consumption threshold (II), the interface to determine the number of in-memory time steps (III), and the button to load the geometry (IV); (V) Interactive display of the tracking graph. Selecting node in either the 3D viewer or the graph display causes the corresponding cell to be highlighted (VI) and its attribute information to be displayed in the info window (VII). The last window (VIII) provides the ability to sub-select segments based on attribute values.

While the overall appearance of the system described below is similar to the one used in [6], we stress that the functionality of the new framework is significantly greater. The system proposed in [6] required separate tools to view segmentations and tracking graphs with no means of interchanging information. Furthermore, the drawing of the tracking graphs was severely limited by the performance of standard SVG-viewers. Finally, the system had no mechanism to incorporate additional statistical information. Instead, this paper presents a fully linked system in which the user can explore the tracking graph, the corresponding segmentation, and the conditional statistics simultaneously with on-demand data loading.

In the following we will describe the different aspects of our interface as well as the algorithms used to implement the various features. When discussing the interface we will use Roman numerals referring to Fig. 9 to illustrate the different components.

6.1 Graph Display

One of our two main windows (V) is dedicated to the display of the tracking graph. Similar to our previous work in [6] we use dot [63] to layout the tracking graphs. However, unlike [6] we no longer rely on external tools to draw the resulting SVG file. Instead, we have developed a fully interactive OpenGL-based display of the graph that provides two key advantages: First, the graph view is now directly linked with the segmentation display and thus can be used to select nodes and or time steps; Second, we can easily apply various color maps to the nodes of the graph representing one additional attribute of each node. To reduce the visual clutter, only the non-valence two nodes of the tracking graph are shown while sequences of valence two nodes are indicated by unbroken arcs. For exploration we typically use the cell volume (represented by the number of vertices within the cell) to highlight larger cells. To display the graph, we load its geometry into OpenGL, which allows us to draw even the largest graphs fully interactively. As evidenced by the accompanying SVG and PDF files, this is a significant improvement over standard tools.

The graph display not only provides a visualization of the graph but the user can also select nodes or arcs. When selecting an arc, the system automatically selects the closest valence two node along this arc. A selection triggers two actions. First, the system loads the merge tree of the corresponding time step and, if desired, a number of neighboring time steps (see below). Since merge trees are comparatively small, the trees are loaded interactively from disk without any caching or other acceleration mechanism. Second, the segment id and all its corresponding attribute information are extracted from the merge tree and displayed in the info window (VII). This linked selection mechanism is a significant improved over the previous system [6], which required the user to view graphs and segmentation in isolation without means to exchange selections or other information.

Note that the current threshold of the merge tree is driven by the segmentation display (slider II), see below, while the graph uses a single fixed threshold. Thus, during selection graph and merge tree can use different thresholds, in which case the system automatically adapts the selection: If the merge tree threshold is smaller (bigger cells) the segment containing the picked one is selected; If the merge tree threshold is larger (smaller cells) the information for the appropriate subsegment is shown. Finally, if the node the user has selected corresponds to any segment currently shown, this segment will be highlighted (VI).

6.2 Segmentation Display

The other main window (I) displays the segmentation and allows the user to vary the threshold (II) and pick the number of in-memory time steps (III). Even though the segmentation is tiny when compared to the original data, parsing roughly 150Mb of binary data into the display data structures currently cannot be performed interactively. Thus, we always load the merge tree information first, providing attribute and hierarchy information. If the user wants to explore a segmentation in more detail, the necessary data is loaded via a button within about a second depending on data size and available hardware. Note, that this performance could likely be improved significantly using compression to reduce I/O time or by preprocessing the segmentations further to become more closely aligned with the display data structures. As mention before, data is cell centered and piece-wise constant. Therefore, we display each vertex of the segmentation as a box to preserve as much of the characteristics of the original data as possible. Individual cells are displayed using one of eleven colors at random, reserving bright-red for highlighted cells. Looking at the segmentation it is important to remember that we use the full 26 neighborhood when computing the merge trees. Thus, even cells touching only at their corners are considered connected.

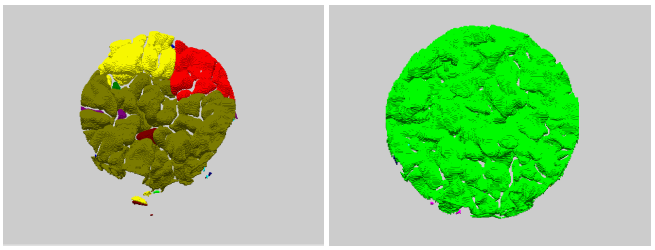
Similar to the graph display, the segmentation view supports selection of individual segments displaying their information in a separate window (VII). Finally, we provide an additional window (VIII) that makes it possible to sub-select segments based on the various attributes. We note that we do not use the attributes directly stored in

the merge tree. Instead, we aggregate attributes by means of pair-wise statistics [62] following the merge tree simplification corresponding to the currently selected threshold. As a consequence, selection is always performed using the attribute values appropriate for the current segmentation. Overall, the system supports to exploring the entire time series of a combustion simulation at arbitrary thresholds and using conditional selection criteria.

7 Results

This paper presents a new visualization and analysis framework to explore one-parameter families of time-dependent segmentations. For the first time it is possible to interactively choose between any of the potential segmentations while exploring additional data attributes. The system is based on a new streaming merge tree algorithm that overcomes the unfavorable scaling of its intellectual ancestor [7] and is much easier to compute than similar segmentations [6, 44]. Furthermore, we have presented a linked-view environment integrating the browsing of large tracking graphs with the interactive visualization of flexible segmentations and coupled to conditional sub-selections based on arbitrary data attributes. Finally, the flexible, one-parameter families of segmentations and their corresponding statistical information enables in depth statistical analysis and parameter studies, which would be infeasible using current techniques. With respect to the application presented here, our framework represents fundamentally new capabilities in studying turbulent flames. Two co-authors of this paper (Marcus Day and John Bell) in particular are actively using the information provided by our system to formulate new hypotheses on the combustion process.

The first significant observation is that the flames in the low-swirl configuration seem to burn in two different modes, see Fig. 1(e). Overall, the burning cells create a bowl shaped structure centered above the burner. To better highlight the center of the flame, the images in this paper, as well as the accompanying movies, show the flame up-side-down looking toward the bottom of the bowl in direction of the fuel stream. Around the center of this bowl, cells appear to behave much like the idealized flames studied in [64]. On the outside, however, the flames burn more chaotically in smaller, irregularly shape regions. The behavior of these *fringe* cells is very unlike that of the idealized flames and it is not yet clear how to model them. Therefore, the initial analysis has focused on the center of the bowl.



(a) (b)

Fig. 10. Burning cells within a cylinder of radius 2.5cm centered in the middle of the data using $2.6 kg_{H_2}/m^3 s$ as fuel consumption cut-off. The view direction is is the direction of the fuel stream. (a) Time step 1500 of the SwirlH2 data; (b) Time step 3000 of the SwirlH2Fast data.

Based on observing the cell structures, a cut-off radius is selected and we extract only the data on the interior of a cylinder with radius 2.5cm centered on the burner, see Fig. 10. The corresponding block of data has dimensions $321 \times 321 \times 251$ compared to the 1024^3 samples of the complete data set. We then compute the hierarchical merge trees and the corresponding segmentations necessary to explore this data in more detail. Studying the segmentations at different thresholds reveals significant differences to the idealized flames of [64, 6]. As can be seen in Fig. 10, the previously used threshold of $2.6 kg_{H_2}/m^3 s$ no longer represents a viable choice. Rather than separating the volume into the cellular burning regions, it defines few very large cells inconsistent with the initial expectation. Instead, for the SwirlH2 data set the segmentations suggest a threshold of around $5 kg_{H_2}/m^3 s$, see Fig. 13, and for the SwirlH2Fast an even higher threshold at around $8 kg_{H_2}/m^3 s$, see Fig. 14. To validate these empirical observations, we compute the weighted cumulative density functions (WCDFs) of the distribution of cell size and compared them to the idealized flames. We also repeated the surface based analysis introduced in [6] to arrive at two sets of distribution functions for each low-swirl experiment, see Fig. 15. As suggested by the visualization, the distributions show a markedly different behavior for lower fuel consumption thresholds. For small thresholds the distributions become exponential indicating a small number of larger cells rather than the logarithmic behavior seen in previous studies. However, as the threshold increases, the distributions continuously change to a logarithmic shape. Furthermore, the area distributions of the 2D analysis changes its characteristic at a significantly lower threshold.

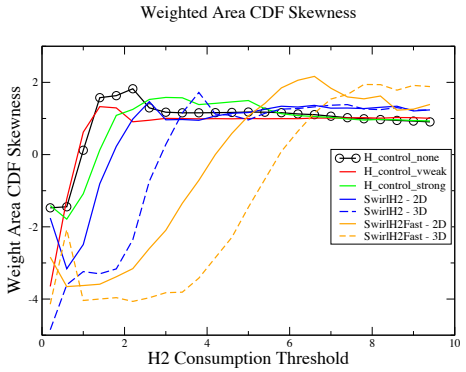


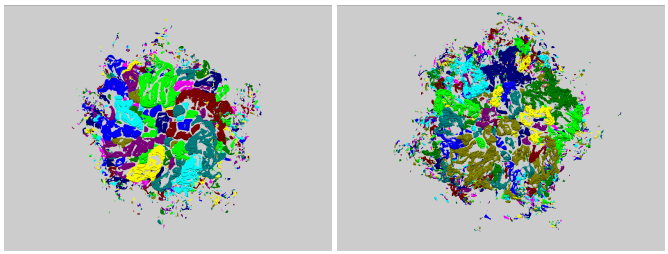
Fig. 11. Skewness plots of the weighted cumulative density functions of the idealized flames analyzed in [64] together with the skewness of the 2D and 3D WCDFs of the SwirlH2 and SwirlH2Fast data. All curves show qualitatively similar shapes switching from exponential WCDFs to logarithmic ones as the fuel consumption threshold is increased. However, for the swirling flames studied here the distributions are noticeably shifted to higher fuel consumption and the 3D WCDFs are shifted further than the 2D WCDFs.

To further quantify this shift, we compute the skewness [65] of the distributions of both idealized flames at different turbulence levels as well as the low-swirl flames presented here. The skewness of an exponential type cumulative density function would be smaller than zero, that of a linear CDF zero, and that of a logarithmic CDF larger than zero. The resulting graphs show several interesting results, see Fig. 11. Clearly the surface based analysis skews the distributions to become logarithmic for smaller thresholds. Furthermore, the graphs for the three-dimensional segmentations validate the visual impression of thresholds around 5.0 and 8.0 for the SwirlH2 and SwirlH2Fast case respectively. Finally, even the idealized flames show exponential WCDFs at very low thresholds something not seen in previous studies.

It is important to point out that each of the data points in Fig. 11 represents a complete analysis of all time steps of the respective data sets for one particular threshold. Thus, creating this plot using traditional techniques would require 168 separate processing runs. Instead, the entire statistical analysis presented here requires only the hierarchical merge trees not the original data. This fact is crucial to allow such extensive studies since accessing the original data would be prohibitively expensive. While seemingly mundane the reduction in disk space necessary to store the "entire" simulation is one of the significant practical benefits of our framework. We store the hierarchy, segmentation, and corresponding index maps of the entire simulation (not the radius based sub-section) using roughly 13GB and 20GB of gzipped files for the SwirlH2 and SwirlH2Fast case, respectively. Given the 3.9 and 4.5 Terabytes of original data this corresponds to a compression of more than two orders of magnitude without sacrificing information. This allows us to visualize and analyze the data on commodity hardware, something infeasible using the original data.

Using the skewness plots of Fig. 11 as a guide we choose 5 (SwirlH2) and 8 (SwirlH2Fast) as thresholds for the tracking and created the corresponding graphs. We use dot [63] to compute a layout and use the resulting svg files for display purposes. Unfortunately, the graphs are too large to be included in this paper. However, we provide these graphs and others at different levels of simplification as additional material, and some are shown in the accompanying movies. As discussed in Section 6 we can use the tracking graphs along side the segmentations to explore an entire time sequence on commodity hardware.

Combining the visual observations of Fig. 13 and 14 with the statistical results shown in Fig. 15 might suggest that the swirling flames behave similar to the idealized flames but at overall higher fuel consumption rates. However, further exploration shows that the results of the spatially restricted data are misleading. Clearly some cells cross the cylindrical center region and thus get cut. One would therefore expect some cells to appear slightly smaller, which could be taken into account during the analysis. However, a more significant problem is that cells can also become disconnected. In fact, segmentations of the entire data show that most cells that appear isolated in the center of the data are connected on the outside of the cylinder, see Fig. 16 and 17. This creates small sets of large cells for even higher thresholds than the plots of Fig. 11 suggest. Only for significantly higher thresholds of 8.0 and 12.0 do the complete flames break apart into smaller components, see Fig. 12. Overall, it appears that the low-swirling flames



(a) (b)

Fig. 12. (a) Burning cells in the SwirlH2 data at time step 1500 using a fuel consumption threshold of $8.0 kg_{H_2}/m^3 s$; (b) Burning cells in the SwirlH2Fast data at threshold $12.0 kg_{H_2}/m^3 s$.

are in a substantially different regime than the idealized flames. Our interactive framework coupled with the data analysis made possible by using augmented hierarchical merge trees has been instrumental in trying to better understand the underlying dynamics controlling the low-swirling flames and has open several new research directions.

All data processing was performed in parallel on an SGI Altix 350, with 32 Itanium-2, 1.4 GHz processors using one processor per time step. The movies were created using a single core of an Apple Macintosh server with eight 3 GHz Intel Xeon processors. The run times for a single representative time step for computing the merge trees and corresponding segmentation are given in Table 1. To better illustrate the cost split between file I/O and computation we report two numbers for each case: First, the time for extracting the raw data and dumping the resulting stream to a file. Second, the time to process this file. Note that the file I/O for the block based output of the complete files is better aligned with the internal AMR data format, which explains the similar running times even though significantly more data is extracted. As discussed before, we use the flexibility of the streaming computation and discard all incoming vertices below a fuel consumption of 0.1. This is a conservative threshold that nevertheless significantly reduces the amount of data that must be processed without loosing any information of interest. Even after discarding all vertices and edges with minimal fuel consumption the remaining mesh for the SwirlH2Fast data set consists of more than five Giga bytes of binary data.

The table also highlights the difficulties with the approach taken in [46]. Requiring the input to be a regular grid and pre-sorting the data incurs a very significant cost overhead. In particular, past experiments show that re-sampling the original AMR-data can take orders of magnitude longer than the analysis itself while requiring non-trivial amounts of disc-space.

	SwirlH2		SwirlH2Fast	
	Center	Complete	Center	Complete
Parallel File I/O	193sec	355sec	210sec	569sec
File Parsing	15sec	205sec	15sec	361sec
Serial Processing	57sec	1067sec	70sec	2684sec
Data Size (raw)	99Mb	N/A	99Mb	N/A
Data Size (filtered)	N/A	3.0Gb	N/A	5.1Gb

Table 1. Run-times and data sizes for the processing of a single representative time step for the SwirlH2 and SwirlH2Fast data set. The parallel file I/O columns report the times to write the raw data to file, the serial processing columns reports the time to process these files. In practice both processes run in parallel which effectively hides the file I/O. For the restrictions to the center we store a regular grid with no connectivity and enforce the cylindrical cut-off using distance based flags. For the complete data we stream a binary representation of vertices, edges, and finalization directly to the merge tree module. The data sizes reported are determined by re-directing this stream to a file.

7.1 Discussion and Future Work

While we can compute the tracking graphs for the full AMR based data including the cells on the fringes, the resulting graphs are difficult to handle. Dot currently does not scale gracefully to these large graphs and creating a layout can take hours or fail all together. Furthermore, assuming a layout is created the resulting graphs are difficult to interpret even after heavy simplification. Currently the graphs, unlike the segmentations, are computed for a static threshold. The data structures contain sufficient information to create graphs efficiently for variable thresholds. However, to view these graphs would require an interactive layout, which is beyond the current state of the art. Thus, new paradigms are needed to handle such graphs potentially involving more sophisticated simplification and hierarchical representations. Nevertheless, we can explore this data even without the graphs as shown in the accompanying movies.

Finally, loading and drawing the geometry becomes noticeably slower for larger data and increasing the resolution by a factor of eight as planned for the future will push the system beyond its current capabilities. However, the rendering code as well as the file I/O is currently entirely un-optimized and contains many opportunities for future improvements. One approach might be to use a compressed file format to reduce file I/O.

8 Conclusions

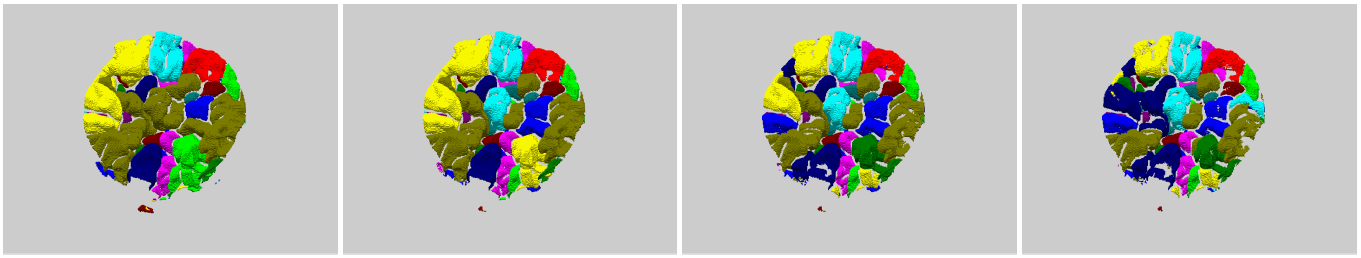
We have presented an interactive framework for exploring and analyzing one-parameter families of segmentations and applied it to study large scale turbulent combustion. Using hierarchical merge trees and their corresponding segmentation we allow to both visualize and post-process entire simulations using pre-processed data orders of magnitude smaller than the original data sets. Providing easy access to correct and comprehensive segmentations including derived attributes has proven to be a powerful tool to better understand turbulent combustion and to form new hypotheses on the underlying physical processes.

Acknowledgements

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-JRNL-447214. This work was supported by:

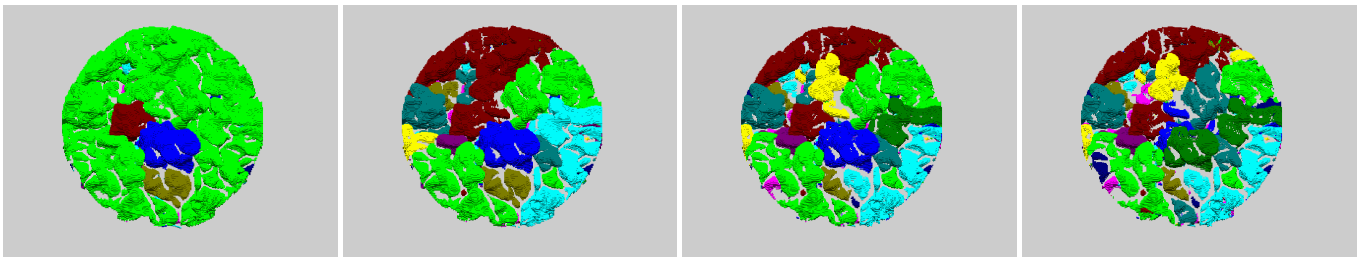
The Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 through the Scientific Discovery through Advanced Computing (SciDAC) program's Visualization and Analytics Center for Enabling Technologies (VACET);

The SciDAC Program of the DOE Office of Mathematics, Information, and Computational Sciences under the U.S. Department of Energy under contract No. DE-AC02-05CH11231. Computational resources have been made available on the Fraklin machine at NERSC



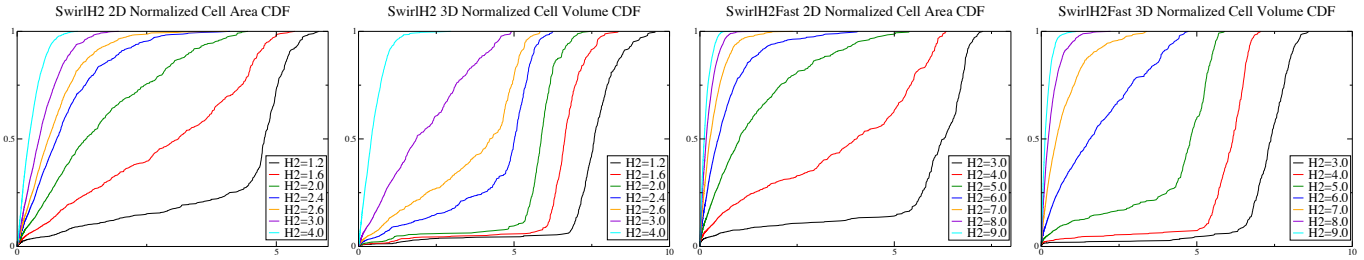
(a) (b) (c) (d)

Fig. 13. Burning cells in the center of the SwirlH2 data at time step 1500 randomly colored using 4.0 (a), 5.0 (b), 6.0 (c), and 7.0 (d) as fuel consumption threshold respectively.



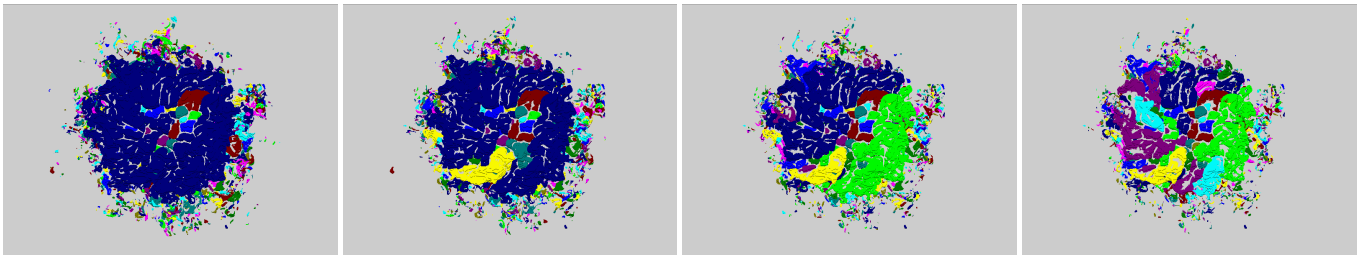
(a) (b) (c) (d)

Fig. 14. Burning cells in the center of the SwirlH2Fast data at time step 3000 randomly colored using 6.0 (a), 7.0 (b), 8.0 (c), and 9.0 (d) as fuel consumption threshold respectively.



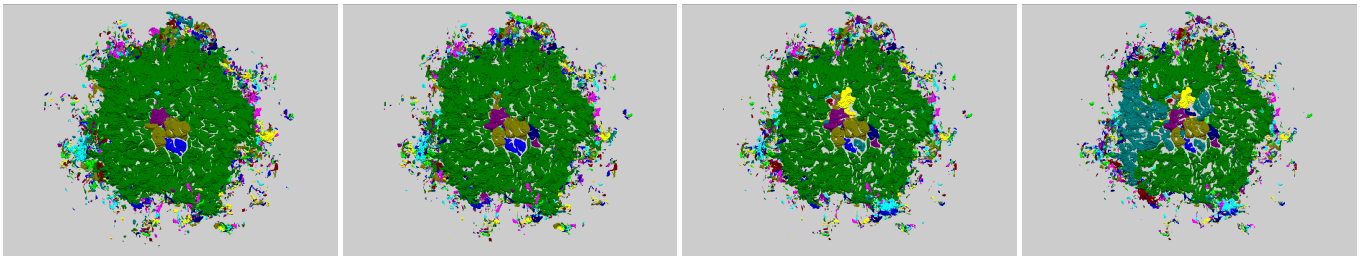
(a) (b) (c) (d)

Fig. 15. Weighted cumulative density functions of the distributions of cell sizes for various thresholds for the surface based analysis of [6] (a),(c) and the volumetric analysis presented here (b),(d) for the SwirlH2 and SwirlH2Fast data set respectively. Unlike the idealized flames of [64] the distributions for lower thresholds are exponential indicating few large cells rather than the expected many small cells.



(a) (b) (c) (d)

Fig. 16. Burning cells in the SwirlH2 data at time step 1500 randomly colored using 4.0 (a), 5.0 (b), 6.0 (c), and 7.0 (d) as fuel consumption threshold, respectively.



(a) (b) (c) (d)

Fig. 17. Burning cells in the SwirlH2Fast data at time step 3000 randomly colored using 6.0 (a), 7.0 (b), 8.0 (c), and 9.0 (d) as fuel consumption threshold, respectively.

as part of an INCITE award and on the Columbia machine at NASA as part of an National Leadership Class System allocation; and

The National Science Foundation (NSF) through the Topology-based Methods for Analysis and Visualization of Noisy Data project.

This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

DISCLAIMER

REFERENCES

[1] W. Lorensen and H. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *Computer Graphics (Proc. SIGGRAGPH '87)*, vol. 21, no. 4, pp. 163–169, July 1987.

[2] G. M. Nielson, "On marching cubes," *IEEE Trans. Vis. Comp. Graph.*, vol. 9, no. 3, pp. 341–351, 2003.

[3] H.-W. S. Yarden Livnat and C. R. Johnson, "A near optimal isosurface extraction algorithm using the span space," *IEEE Transactions on Visualization and Computer Graphics*, vol. 2, no. 1, pp. 73–84, 1996.

[4] B. Bedat and R. K. Cheng, "Experimental study of premixed flames in intense isotropic turbulence," *Combust. Flame*, vol. 100, pp. 485–494, 1995.

[5] R. K. Cheng, "Velocity and scalar characteristics of premixed turbulent flames stabilized by weak swirl," *Combust. Flame*, vol. 101, no. 1-2, pp. 1–14, 1995.

[6] P.-T. Bremer, G. Weber, V. Pascucci, M. Day, and J. Bell, "Analyzing and tracking burning structures in lean premixed hydrogen flames," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 2, pp. 248–260, 2010.

[7] V. Pascucci, G. Scorzelli, P.-T. Bremer, and A. Mascarenhas, "Robust online computation of Reeb graphs: Simplicity and speed," *ACM Trans. on Graph.*, vol. 26, no. 3, pp. 58.1–58.9, 2007.

[8] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch, "The state of the art in flow visualisation: Feature extraction and tracking," *Computer Graphics Forum*, vol. 22, no. 4, pp. 775–792, 2003.

[9] I. Fujishiro, Y. Maeda, and H. Sato, "Interval volume: a solid fitting technique for volumetric data display and analysis," in *Proc. IEEE Visualization 1995*, 1995, pp. 151–158.

[10] K. Stockinger, J. Shalf, K. Wu, and E. W. Bethel, "Query-driven visualization of large data sets," in *Proc. IEEE Visualization 2005*, 2005, pp. 167–174.

[11] L. Gosink, J. C. Anderson, E. W. Bethel, and K. I. Joy, "Variable interactions in query-driven visualization," *IEEE Trans. Vis. Comp. Graph.*, vol. 13, no. 6, pp. 1400–1407, 2007.

[12] A. Mascarenhas and J. Snoeyink, *Isocontour based Visualization of Time-varying Scalar Fields*. Springer Verlag, 2009.

[13] R. Samtaney, D. Silver, N. Zabusky, and J. Cao, "Visualizing features and tracking their evolution," *IEEE Computer*, vol. 27, no. 7, pp. 20–27, 1994.

[14] D. Silver and X. Wang, "Tracking and visualizing turbulent 3d features," *IEEE Transactions on Visualization and Computer Graphics*, vol. 3, no. 2, pp. 129–141, 1997.

[15] ——, "Tracking scalar features in unstructured datasets," in *Proc. IEEE Visualization '98*. IEEE Computer Society Press, 1998, pp. 79–86.

[16] D. Laney, P.-T. Bremer, A. Mascarenhas, P. Miller, and V. Pascucci, "Understanding the structure of the turbulent mixing layer in hydrodynamic instabilities," *IEEE Trans. Vis. Comp. Graph.*, vol. 12, no. 5, pp. 1052–1060, 2006.

[17] G. Ji, H.-W. Shen, and R. Wegner, "Volume tracking using higher dimensional isocontouring," in *Proc. IEEE Visualization '03*. IEEE Computer Society, 2003, pp. 209–216.

[18] G. Ji and H.-W. Shen, "Efficient isosurface tracking using precomputed correspondence table," in *Proc. IEEE/Eurographics Symposium Visualization '04*, 2004, pp. 283–292.

[19] G. Weber, P.-T. Bremer, J. Bell, M. Day, and V. Pascucci, *Feature Tracking Using Reeb graphs*, ser. Mathematics and Visualization. Springer, 2010, in press.

[20] H. Edelsbrunner, J. Harer, A. Mascarenhas, V. Pascuccii, and J. Snoeyink, "Time-varying Reeb graphs for continuous space-time data," *Computational Geometry*, vol. 41, no. 3, pp. 149–166, 2008.

[21] A. Szymczak, "Subdomain-aware contour trees and contour tree evolution in time-dependent scalar fields," in *Proc. Shape Modeling International (SMI) '05*. IEEE Computer Society, 2005, pp. 136–144.

[22] H. Edelsbrunner and J. Harer, "Jacobi sets of multiple Morse functions," in *Foundations of Computational Mathematics, Minneapolis 2002*, F. Cucker, R. DeVore, P. Olver, and E. Sueli, Eds. Cambridge Univ. Press, England, 2002, pp. 37–57.

[23] B.-S. Sohn and C. Bajaj, "Time-varying contour topology," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 1, pp. 14–25, 2006.

[24] P.-T. Bremer, E. Brings, M. Duchaineau, A. Gyulassy, D. Laney, A. Mascarenhas, and V. Pascucci, "Topological feature extraction and tracking," *Proceedings of SciDAC 2007 - Scientific Discovery Through Advanced Computing*, vol. 78, pp. 012 032 (5pp), Journal of Physics Conference Series, June 2007.

[25] M. Morse, "Calculus of variations in the large," *Amer. Math. Soc. Colloquium Publications*, vol. 18, 1934.

[26] J. Milnor, *Morse Theory*. New Jersey: Princeton University Press, 1963.

[27] G. Reeb, "Sur les points singuliers d'une forme de pfaff completement intergrable ou d'une fonction numerique [on the singular points of a complete integral pfaff form or of a numerical function]," *Comptes Rendus Acad.Science Paris*, vol. 222, pp. 847–849, 1946.

[28] R. L. Boyell and H. Ruston, "Hybrid techniques for real-time radar simulation," in *Proc. 1963 Fall Joint Comp. Conf.*, 1963, pp. 445–458.

[29] S. Takahashi, T. Ikeda, Y. Shinigawa, T. L. Kunii, and M. Ueda, "Algorithms for extracting correct critical points and constructing topological graphs from discrete geographical elevation data," in *Proc. Eurographics '95*, Sep. 1995, pp. C–181–C–192.

[30] T. Itoh and K. Koyamada, "Automatic isosurface propagation using an extrema graph and sorted boundary cell lists," *IEEE Trans. Vis. and Comp. Graph.*, vol. 1, no. 4, pp. 319–327, 1995.

[31] M. J. van Kreveld, R. van Oostrum, C. L. Bajaj, V. Pascucci, and D. Schikore, "Contour trees and small seed sets for isosurface traversal," in *Symposium on Computational Geometry*, 1997, pp. 212–220.

[32] H. Carr, J. Snoeyink, and U. Axen, "Computing contour trees in all dimensions," *Comput. Geom. Theory Appl.*, vol. 24, no. 3, pp. 75–94, 2003.

[33] V. Pascucci and K. Cole-McLaughlin, "Parallel computation of the topology of level sets," *Algorithmica*, vol. 38, no. 1, pp. 249–268, Oct. 2003.

[34] S. Takahashi, G. M. Nielson, Y. Takeshima, and I. Fujishiro, "Topological volume skeletonization using adaptive tetrahedrization," in *Proc. of Geometric Modeling and Processing 2004*, 2004, pp. 227–236.

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

[36] H. Edelsbrunner, J. Harer, and A. Zomorodian, "Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds," *Discrete Comput. Geom.*, vol. 30, pp. 87–107, 2003.

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

[38] A. Gyulassy, V. Natarajan, V. Pascucci, P.-T. Bremer, and B. Hamann, "Topology-based simplification for feature extraction from 3D scalar fields," in *Proc. IEEE Visualization 2005*, 2005, pp. 535–542.

[39] ——, "Topology-based simplification for feature extraction from 3D scalar fields," *IEEE Transactions on Visualization and Computer Graphics (Proc. IEEE Vis.)*, vol. 12, no. 4, pp. 474–484, 2006.

[40] A. Gyulassy, V. Natarajan, V. Pascucci, and B. Hamann, "Efficient computation of Morse-Smale complexes for three-dimensional scalar functions," *IEEE Transactions on Visualization and Computer Graphics (Proc. IEEE Vis)*, vol. 13, no. 6, pp. 1440–1447, 2007.

[41] H. Carr and J. Snoeyink, "Path seeds and flexible isosurfaces using topology for exploratory visualization," in *Proc. VisSym '03*, 2003, pp. 49–58.

[42] H. Carr, J. Snoeyink, and M. van de Panne, "Simplifying flexible isosurfaces using local geometric measures," in *Proc. IEEE Visualization 2004*, 2004, pp. 497–504.

[43] G. H. Weber, S. E. Dillard, H. Carr, V. Pascucci, and B. Hamann, "Topology-controlled volume rendering," *IEEE Trans. Vis. Comp. Graph.*, vol. 12, no. 2, pp. 330–341, 2007.

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

[45] S. Takahashi, I. Fujishiro, and M. Okada, "Applying manifold learning to plotting approximate contour trees," *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization/Information Visualization 2009)*, vol. 15, no. 6, pp. 1185–1192, 2009.

[46] A. Mascarenhas, R. W. Grout, P.-T. Bremer, E. R. Hawkes, V. Pascucci, and J. Chen, *Topological feature extraction for comparison of terascale combustion simulation data*, ser. Mathematics and Visualization. Springer, 2010, to appear.

[47] P.-T. Bremer, G. H. Weber, J. Tierny, V. Pascucci, M. Day, and J. B. Bell, "A topological framework for the interactive exploration of large scale turbulent combustion," in *Proc. IEEE International Conference on e-Science*. IEEE, December 2009, pp. 247–254.

[48] M. Q. Wang Baldonado, A. Woodruff, and A. Kuchinsky, "Guidelines for using multiple views in information visualization," in *AVI '00: Proc. of the working conference on Advanced visual interfaces*, 2000, pp. 110–119.

[49] C. Henze, "Feature detection in linked derived spaces," in *Proc. IEEE Visualization '98*, 1998, pp. 87–94.

[50] D. L. Gresh, B. E. Rogowitz, R. L. Winslow, D. F. Scollan, and C. K. Yung, "WEAVE: A system for visually linking 3-d and statistical visualizations, applied to cardiac simulation and measurement data," in *Proc. IEEE Visualization 2000*, 2000, pp. 489–492.

[51] H. Doleisch, M. Gasser, and H. Hauser, "Interactive feature specification for focus+context visualization of complex simulation data," in *Data Visualization 2003*, 2003, pp. 239–248.

[52] I. Fujishiro, R. Otsuka, S. Takahashi, and Y. Takeshima, "T-map: A topological approach to visual exploration of time-varying volume data," in *Proc. of 6th International Symposium on High Performance Computing*, 2008, pp. 176–190.

[53] F. A. Williams, *Combustion Theory*, 2nd ed. Westview Press, 1994.

[54] T. Poinsot and D. Veynante, *Theoretical and Numerical Combustion*, 2nd ed. R.T. Edwards, 2005.

[55] J. Bell, M. Day, A. Almgren, M. Lijewski, C. Rendleman, R. Cheng, and I. Shepherd, "Simulation of lean premixed turbulent combustion," *SciDAC 2006 (Journal of Physics: Conference Series)*, vol. 46, pp. 1–15, 2006.

[56] P. Peterson, J. Olofsson, C. Brackman, H. Seyfried, J. Zetterberg, M. Richter, M. Alden, M. Linne, R. Cheng, A. Nauert, D. Geyer, and A. Dreizler, "Simultaneous PIV/OH PLIF, Rayleigh thermometry/OH PLIF and stereo PIV measurements in a low-swirl flame," *Appl. Opt*, vol. 46, pp. 3928–3936, 2007.

[57] K. Nogenmyr, P. Peterson, X. Bai, A. Nauert, J. Olofsson, C. Brackman, H. Seyfried, Z.-S. Zetterberg, J. Li, M. Richter, A. Dreizler, M. Linne, and Alden, "Large eddy simulation and experiments of stratified lean premixed methane/air turbulent flames," *Proc. Combust. Inst*, vol. 31, pp. 1467–1475, 2007, *submitted to LACSEA Feature Issue*.

[58] M. Mansour and Y.-C. Chen, "Stability characteristics and flame structure of low swirl burner," *Experimental Thermal and Fluid Science*, vol. 32, no. 7, pp. 1390–1395, Jul. 2008, fifth mediterranean combustion symposium.

[59] M. S. Day and J. B. Bell, "Numerical simulation of laminar reacting flows with complex chemistry," *Combust. Theory Modelling*, vol. 4, pp. 535–556, 2000.

[60] V. Pascucci, K. Cole-McLaughlin, and G. Scorzelli, "Multi-resolution computation and presentation of contour trees," LLNL, Tech. Rep. UCRL-PROC-208680, 2004.

[61] H. Carr and J. Snoeyink, "Representing interpolant topology for contour tree computation," in *Topology-Based Methods in Visualization II*, ser. Mathematics and Visualization, H.-C. Hege, K. Polthier, and G. Scheuermann, Eds. Springer Berlin Heidelberg, 2009, pp. 59–73.

[62] J. Bennett, R. Grout, P. Pébay, D. Roe, and D. Thompson, "Numerically stable, single-pass, parallel statistics algorithms," in *Proceedings of IEEE Cluster 2009*, 2009, p. to appear.

[63] E. Koutsofios and S. North, "Drawing graphs with dot," AT&T Bell Laboratories, Murray Hill, NJ, Tech. Rep. 910904-59113-08TM, 1991.

[64] 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*, vol. 156, pp. 1035–1045, 2009.

[65] J. L. Devore, *Probability and Statistics for Engineering and the Sciences*. Belmont, CA: Brooks/Cole - Thomson Learning, 2004.