Eurographics Symposium on Parallel Graphics and Visualization (2015)
C. Dachsbacher, P. Navrátil (Editors)

Contour Tree Depth Images For Large Data Visualization

T. Biedert and C. Garth

University of Kaiserslautern, Germany

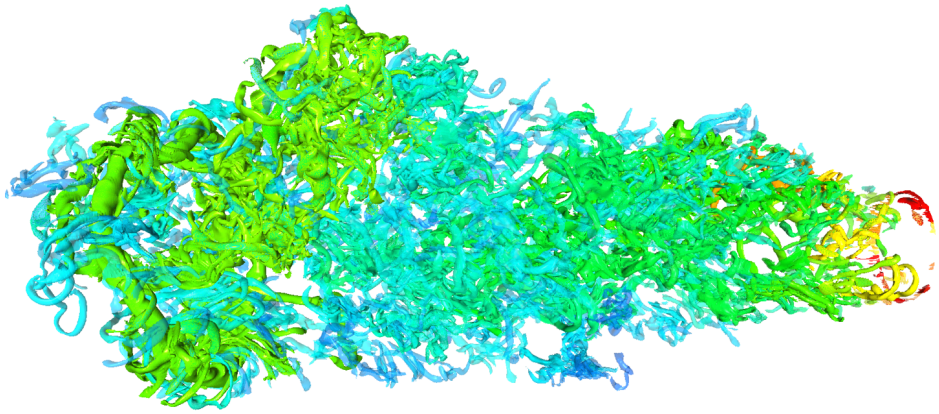


Figure 1: Automatic segmentation of the turbulent *jet5* λ_2 data set into 1024 branches based on branch persistence.

Abstract

High-fidelity simulation models on large-scale parallel computer systems can produce data at high computational throughput, but modern architectural trade-offs make full persistent storage to the slow I/O subsystem prohibitively costly with respect to time. We demonstrate the feasibility and potential of combining in situ topological contour tree analysis and compact image-based data representation to address this problem. Our experiments show significant reductions in storage requirements using topology-guided layered depth imaging, while preserving flexibility for explorative visualization and analysis. Our approach represents an effective and easy-to-control trade-off between storage overhead and visualization fidelity for large data visualization.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types

1. Introduction

At the current scale of computational capability provided by large-scale parallel computer architectures such as commodity clusters and modern supercomputers, high-fidelity computational simulation models have assumed a significant role in scientific research and engineering applications. However, this increased amount of computation has incurred architectural trade-offs. While arithmetic capacity and in-core memory have grown at a tremendous rate, I/O subsystems have not been able to keep abreast in relative bandwidth [CPA*10]. As a consequence, numerical data produced during typical simulations cannot be persistently stored, e.g. to hard drives, in its entirety; a lack of available I/O bandwidth would make this prohibitively costly with respect to time.

Visualization of large-scale simulation output thus has to rely on a number of different strategies to facilitate meaningful analysis in reasonable time frames. Multi-resolution schemes represent data on varying scales of resolutions and have a long standing tradition in this setting. They enable

an essential compromise between fidelity and accuracy of visualization results on one hand, and computation and I/O bandwidth expended on the other. Among the large set of available methods, topological techniques such as the *contour tree* stand out because they are able to provide meaningful simplification for scalar fields.

A further smart approach to reduce the volume of data generated from simulations while preserving a significant amount of analysis flexibility was recently introduced by Ahrens et al. [AJO*14]. The underlying idea is to generate in situ an image database that stores multiple layers of predefined visualization renderings. The layers can then be composited in post processing depending on specific demands by the scientist conducting analysis.

In this context, the intent of this paper is to study the combination of in situ topological analysis with the image-based approach of Ahrens et al. Based on in situ contour tree analysis and simplification, we obtain a segmented representation of scalar fields contained in the simulation data at every time step. A rendering of this segmentation is then generated describing all components visible in every pixel (similar to an A-buffer), and stored together with the simplified contour tree. These ingredients can then be used in post-analysis to flexibly select specific subsets of the segmentation, after further simplification if required.

The overall intent of this paper is to investigate possible advantages of such an approach for the visualization of large-scale data. Specifically, after a brief review of relevant prior work (Section 2), we make the following contributions:

- In Sections 3 through 7, we describe a system to combine in situ contour tree analysis, simplification, and image-based representation to facilitate reduced I/O requirements while preserving flexibility in visualization.
- We conduct several experiments to quantify the I/O savings possible from such an approach, and describe results and analysis in Section 8.
- We anticipate that many enhancements and improvements are possible, and discuss a number of such opportunities in Section 9.

Our contribution is intended as a baseline demonstration of the feasibility and potential of the combination of topological analysis and image-based representation in large-scale in situ scenarios.

2. Related Work

A classic use case of topology in scalar field visualization is isosurface extraction, where typically several topological properties such as the number of connected components or the genus of the isosurface, i.e., the number of independent tunnels, are of central interest. Based on Morse theory, showing that topological changes in scalar fields defined on manifolds happen only at distinct critical points, Reeb graphs capture the topological evolution of individual contours using these critical points and their relationships. The efficient construction of Reeb graphs in general is still an active field of study [DN09]. However, for simply connected domains, the Reeb graph is always a tree structure [BR63], called *contour tree*, which is algorithmically computable for tetrahedral [CSA00] and hexahedral [PCM04] meshes. Since contour trees can become hard to understand due to high complexity, Weber et al. [WBP07] introduced *topological landscapes*, a visual metaphor for contour trees by creating a representative terrain with the same topological structure as a given contour tree, which can be further extended to reflect the geometric proximity of the features represented therein [BWM*12] or hierarchically used for topology exploration [DBW*12].

Topological techniques have proven highly valuable for the analysis, visualization and exploration of scientific data. Bajaj et al. [BPS97] introduced the *contour spectrum*, an interface providing the contour tree and additional properties such as area and enclosed volume alongside isosurface visualization. Fujishiro et al. detected significant isovalues automatically using the contour tree for transfer function design [FTAT00]. Weber et al. based scalar field exploration on the detection of critical points and critical regions [WSH03]. Van Kreveld et al. [vKvOB*97] performed seeded isosurface extraction based on the contour tree, which was extended by Carr and Snoeyink to use the contour tree as a visual index for a volume data set and identify all contours for a given isovalue [CS03]. Takahashi et al. [TFT05] employed interval volumes to visualize regions of uniform topology, providing means to examine internal structures by peeling away top layers. Takashima et al. [TTFN05] further investigated the idea of peeling off layers by using topological information such as isosurface inclusion level in multi-dimensional transfer function design.

Noise in data sets can lead to large numbers of irrelevant critical points, complicating feature-driven exploration based on topology. Topological simplification eliminates insignificant features by *cancellation*, i.e., removing irrelevant pairs of critical points. The *Volume skeleton tree* [TTF04] and the *Morse-Smale complex* [EHNP03,BEHP04,GNP*05] are two topological structures widely used for scalar topological simplification besides the contour tree. However, our work relies on the simplification approaches introduced by Carr et al. [CSvdP04] and Pascucci et al. [PCMS04]. Carr et al. [CSvdP04] used leaf pruning and node collapse operations for contour tree simplification. Removing a leaf and the arc incident to the leaf from the contour tree discards the corresponding contours from further consideration when using seed-based contour extraction. Pascucci et al. [PCMS04] introduced the *branch decomposition* of a contour tree, which can be interpreted as a hierarchy of contour tree simplifications. Since a branch is defined by a monotone path connecting a saddle and an extremum vertex, discarding a branch is equivalent to the topological cancellation of the respec-

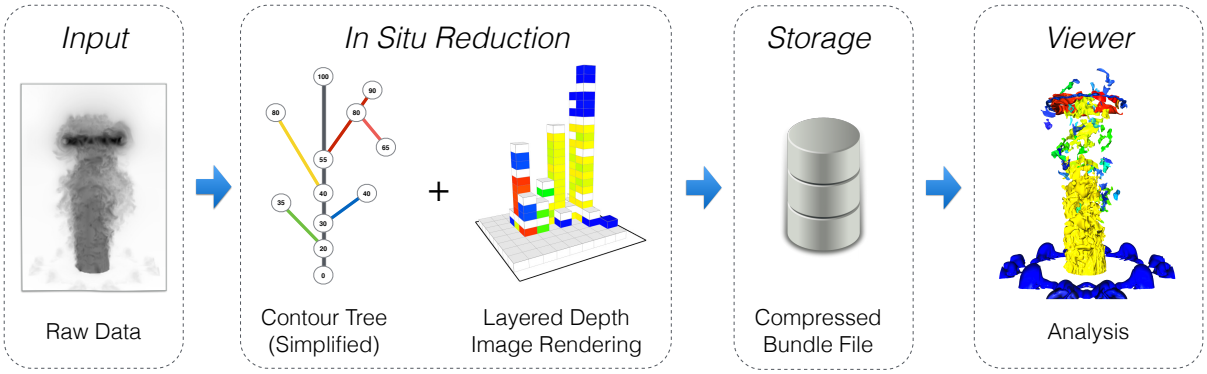


Figure 2: Conceptual architectural overview of our contribution, providing a framework for the flexible exploration of in-situ generated compressed renderings. Volume data is segmented into topological regions based on its contour tree. After automatic filtration, the segments imposed by the simplified contour tree are intersected by a ray casting rendering approach. All resulting fragments form a layered depth image, which is compactly stored combined with the contour tree and of significantly smaller size than the original data.

tive two critical points. Our work is inspired by Weber et al. [WDC*07], who used the branch decomposition for the segmentation of a volume into regions of equivalent contour topology and applying separate transfer functions to each region. Carr et al. [CSvdP10] also used the simplified contour tree as an interface for exploratory visualization.

Bremer et al. studied the application of topological methods to large data scenarios, defining feature identification by thresholding isosurfaces in terms of the Morse complex and representing the complete evolution of all features over time in tracking graphs [BWP*10]. Also, Bremer et al. used hierarchical merge trees as a compact feature representation reducing data storage [BWT*11]. Thompson et al. [TLB*11] introduced *hixels* as a new compact representation of large scalar data, storing a histogram of values for each sample point of the domain, thereby trading off data size and complexity for scalar value uncertainty. Landge et al. used segmented merge trees to encode a wide range of threshold based features to obtain a reduced data representation while maintaining post-processing flexibilities [LPG*14]. More recently, also image-based approaches to large data storage and visualization emerged. Tikhonova et al. [TCM10] presented the idea of using proxy images for interactive exploration without accessing the original 3D data. View changes, transfer function exploration, and relighting are handled in proxy image space only. Ahrens et al. [AJO*14] generated an in situ image database, storing multiple layers of predefined visualization renderings, which can then be composited in post processing. The central idea, which is also key to our work, is to achieve a massive data reduction when storing the simulation output of large-scale numerical simulations, while preserving visualization fidelity and flexibility for future post-processing. Frey et al. presented an interesting novel scheme for progressive rendering which helps to achieve a fluent interactive visualization of large data at high frame rates [FSME14].

Building on this rich methodological foundation, we investigate the feasibility and potential of combining in situ topological analysis and image-based representation to tackle the problems in permanent data storage of high-fidelity simulation results, caused by the modern architectural trade-offs in large-scale parallel computer systems.

3. Method Overview

We aim to achieve substantial reductions in data size by leveraging topological simplification and compact image-based storage. Our approach essentially consists of two components: an image *rendering library* and an interactive *viewer application*. A conceptual overview of the central steps in the processing workflow is depicted in Figure 2.

The rendering library is directly includable into simulation code and targets high-performance cluster environments or workstations. Simulation output data is segmented into topological regions based on its contour tree, which in turn are intersected by a ray casting rendering approach. However, instead of only determining the fragment closest to the camera, we store each intersection together with a set of local properties in a linked fragment list, similar to an A-buffer [Car84]. These properties include the element in the contour tree corresponding to the intersected segment and further additional attributes used for rendering such as the normal or ray depth at the intersection. The fragment lists combined with the contour tree are stored in a proprietary binary file format, which is considerably smaller in size than the original simulation data, yet provides enough flexibility for subsequent data exploration.

Once compressed layered depth image data has been

generated, it represents a significant size reduction of the original input data and can be explored in the interactive viewer application running on the user's desktop computer. By interactively modifying visual properties of the regions imposed by the contour tree or applying further filtering schemes, the user can control which topological regions are displayed.

4. Segmentation and Filtering

Given a scalar field defined on a regular grid as input data, we construct the contour tree using the sweep and merge algorithm by [CSA00], where the split and join sweep phases are executed in parallel. Afterwards, the so-called *branch decomposition* is computed using a variant of [PCMS04]. The branch decomposition can be interpreted as a hierarchy of contour tree simplifications. Since a branch is defined by a monotone path connecting a saddle and an extremum vertex, discarding a branch is equivalent to the topological cancellation of the respective two critical points. The output of the above algorithm is a tree structure representing the branch decomposition of the contour tree and a mapping of vertices to their corresponding branch in the decomposition. Notably, the latter is used for rendering only, whereas the branch decomposition is stored in the final image, with each branch being characterized by index, volume and critical value pair.

Filtering is a central concept inherent to the hierarchical branch decomposition structure. In our context, if a branch is to be discarded, all vertices belonging to this branch are reassigned to the closest unfiltered parent branch. We employ this technique in two ways. First, the initial branch decomposition used for rendering can be controlled by a user-provided maximum number of branches. This is a crucial step to achieve significant data reduction at in situ time, as the initial unfiltered branch decomposition consists of numerous branches which are irrelevant for the later visualization, either due to negligible importance or being caused by noise. Second, in the interactive viewer application, the user can apply several consecutive filtering steps in order to simplify the layered depth image visualization.

Besides manual branch selection, the library provides means to automatically identify branches of interest. Automatic branch selection can be either done by range, i.e., select all branches whose critical values intersect a given input range, or by sorting, i.e., sort all branches either by persistence or volume and pick the first k branches which fulfill a given minimum persistence or volume threshold. Notably, if the sorting criterion of two branches is equal, they are sorted by their depth in the branch decomposition tree, ensuring child branches are discarded before their parent. Once expendable branches are identified, filtering is performed recursively. If a branch is flagged for discard, also all of its children are discarded. Otherwise they are individually checked. However, other filtering criteria are certainly possible.

The consequences of performing a filtering step depend on whether it is done for in situ data reduction or in the viewer application. Automatic filtering after the construction of the initial branch decomposition prior to rendering only requires an update to the vertex-to-branch mapping. However, when applying additional user-controlled filtering in the viewer application, the branch indices of all fragments in all depth images linked to this contour tree need to be updated. Furthermore, since branch filtering can geometrically lead to the merging of neighboring topological regions, replacing branch indices in the linked fragment lists can produce duplicate intersections which need to be eliminated. In this case, we only keep the one closest to the camera.

5. Depth Image Rendering

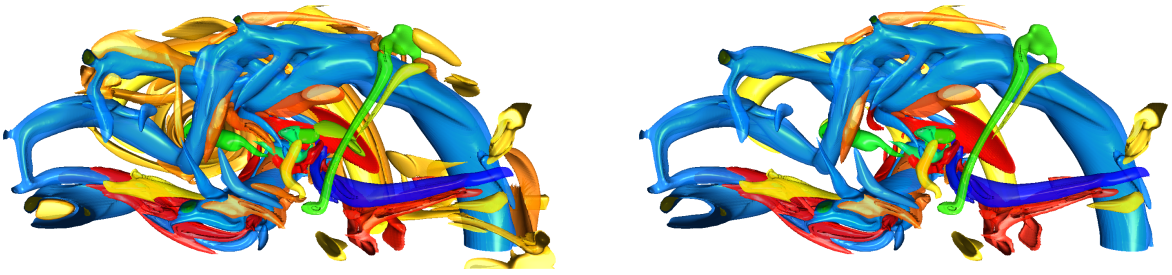
After the contour tree for a single simulation time step has been constructed, multiple layered depth image renderings can be computed based on its branch decomposition. Thus, given a scalar field defined on a regular grid, the vertex-to-branch mapping, camera position and orientation, resolution and an optional maximum number of depth layers, we perform a concurrent ray casting procedure for image generation.

5.1. Segment Intersection

The goal is to record all intersections of rays with the boundaries of the topological regions defined by the contour tree. Assuming a sequence of sample positions along the ray, we principally need to determine the branch index at each location and detect an intersection if the branch index at the current candidate location is different from the branch index at the previous accepted intersection. However, while we want to achieve high intersection precision, performing naïve uniform sampling with sufficiently small step size along the complete ray obviously suffers from bad performance in large homogeneous segments.

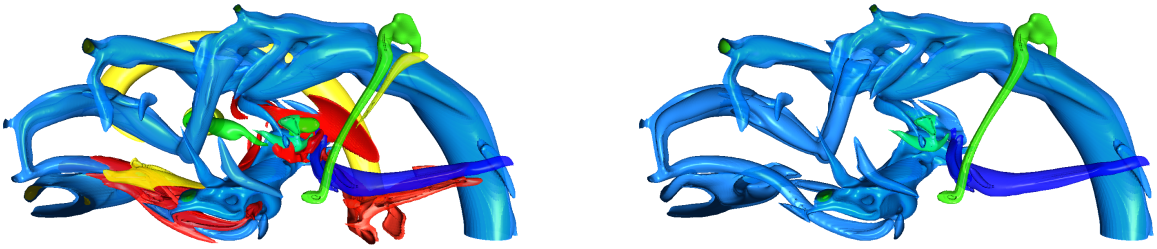
Rather, our sampling algorithm traverses data voxels following a three-dimensional Bresenham approach. For each cell encountered, we check if all eight corner vertices belong to the same branch, in which case we can trivially use this respective branch index and return the entering intersection of the ray with the voxel boundary as intersection candidate. Otherwise, there are potential intersections with topological segment boundaries within the cell, which we approximate by a local uniform sampling scheme restricted to the cell volume.

At each sample position, we need to determine the topological segment containing the given location, i.e., determine the respective branch index. We follow the approach presented in [WDC*07], which is based on the relation that monotone paths in the scalar field always map to monotone paths in the corresponding contour tree and vice versa. First, the unique monotone path going through the given sample



(a) 90 branches

(b) 50 branches



(c) 30 branches

(d) 10 branches

Figure 3: Incremental simplification of the central turbulence in the *plate* λ_2 data set by sorted branch persistence, enabling a flexible and topologically-guided exploration of vortex core structures.

position and two cell corners is determined by exploiting the linearity of the trilinear interpolant within the cell along axis-oriented lines. We then follow the monotone path in the contour tree until the branch containing the data value at the sample position is found. Eventually, for each detected fragment, the associated branch index and additional optional parameters such as gradient of the scalar field or depth value are stored.

Notably, the central idea of this proof-of-concept work is independent of the employed data sampling scheme, providing opportunities for performance optimization. Also, together with an appropriate intersection detection scheme, our approach is easily applicable to non-regular data.

5.2. GPU Acceleration

We have implemented the above layered depth image rendering algorithm both as a CPU- and GPU-version. While the multi-threaded CPU version of our library targets rendering of large data sets in traditional cluster environments without accelerator cards, we have additionally implemented GPU hardware acceleration as a proof of concept of our highly parallelizable algorithm.

As described above, rendering by ray casting creates a list of fragments for each ray, representing the visual properties for all intersections with topological segments it encounters. Our implementation is conceptionally similar to per-pixel linked lists [YHGT10], a recent technique in computer graphics for hardware-accelerated order-independent transparency [MCTB12] based on the classic A-buffer approach [Car84]. For each pixel, a linked list of fragments keeps track of the intersections encountered during traversal. The respective fragment properties themselves are stored in a global shared fragment pool.

Based on OpenGL 4.3 [SA13], our implementation relies on shader storage buffer objects and image load/store extensions. Input volume data and the vertex-to-branch mapping originating from the branch decomposition of the contour tree are stored in uniform volume samplers. Per-pixel information, i.e., the head node of the linked fragment list and the number of nodes in the list, is stored in two uimage2D samplers in r32ui layout, sized at the requested rendering resolution. A shader storage buffer object holds the global fragment pool, where each fragment stores its properties and a link to the next fragment. An atomic counter buffer object is used to keep track of the next free fragment in the global fragment pool. Whenever a newly generated fragment is to be stored in the pool, the counter is incremented, the head node is exchanged and the number of fragments in the list is increased, using the respective atomic buffer and image operations. In order to maintain linked list consistency, the next node of the new fragment is set to the previous head node of the pixel's linked fragment list.

Fast parallel ray generation is achieved by first rendering

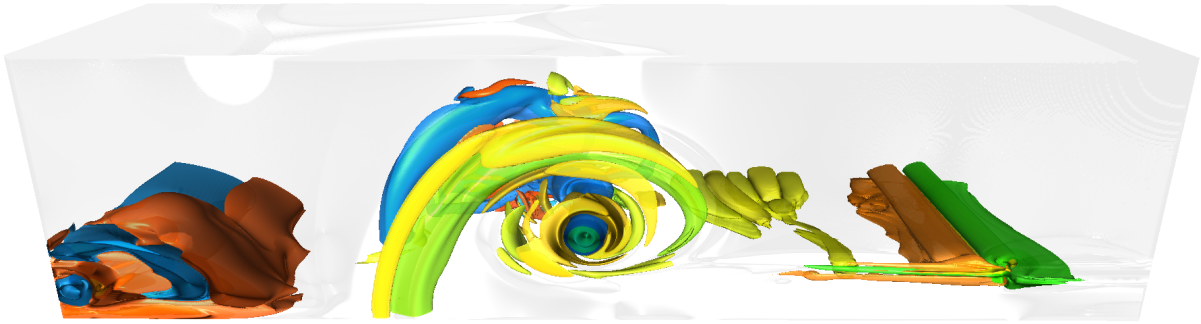


Figure 4: Automatic segmentation of the *plate* λ_2 data set into 100 branches sorted by branch volume. An HSV color scale has been applied to all branches, where for each branch the average scalar branch value was approximated by the mean of saddle and extremum values stored with the branch.

only the back faces of the data volume cube to an offscreen framebuffer and storing the generated interpolated fragment position, i.e., ray exit point of this pixel, in the respective framebuffer's color attachment. In the next pass, the same procedure is applied to the front faces only, yielding the ray starting positions.

In summary, our GPU-based rendering procedure consists of four phases, all of which are executed highly parallel by hardware:

1. Draw back faces of data cube to offscreen framebuffer to obtain ray exit locations.
2. Draw fullscreen quad clearing linked fragment lists, i.e., reset head nodes and node counts in per-pixel samplers.
3. Draw front faces of data cube to obtain ray entry locations.
4. Perform ray casting and store intersections in linked fragment lists.

Step 3 and 4 are executed in a single shader pass. After rendering, the contents of the global fragment pool and the per-pixel samplers are read back to host memory.

6. Storage

Reducing data storage requirements while maintaining flexibility for interactive exploration of results is central to our contribution. Thus, compactly storing the output of our rendering algorithm is crucial. For a given (filtered) contour tree, multiple depth renderings from different perspectives and resolutions can be generated. These bundles are stored in a single file in HDF5 format with zlib deflate compression enabled at level 6, which we found to be a good compromise between size and speed.

The branch decomposition of the contour tree is serialized recursively in depth-first manner, where for each branch we store index, saddle value, extremum value, and volume. However, with only a few kilobytes, the contribution of the branch decomposition to the final file size is negligible. Clearly, the majority of storage is consumed by the depth images, which thus require more optimization w.r.t. memory consumption. We need to store for each pixel a list of fragments, where a single fragment consists of its associated branch index and additional attributes such as normal. While during rendering fragments are stored and manipulated as linked lists distributed across memory and potentially shared by rendering threads, this data can be compactly reorganized for permanent storage. We remove the next-pointers of the linked lists by rearranging fragment lists as contiguous blocks in a single large layout, where each pixel of the final layered depth image only stores the starting offset and the number of fragments.

Since the largest offset, the maximum number of fragments per pixel and the maximum branch index (i.e. total number of branches in the branch decomposition) are known, primitive data types used for storing the respective values can be intelligently chosen. As an example, if we can restrict the total number of branches to 256, this allows branch indices to be stored in single bytes, yet provides sufficient segmentation detail in many scenarios.

While storing the branch index of each fragment is mandatory, additional attributes are optional and depend on the intended visualization. In our studies, we additionally compactly store the normal at each intersection in two bytes using a spheremap transformation [Pra10]. Also, we store the intrinsic and extrensic camera parameters used for image generation, serving as reference for lighting and shading in the viewer.

7. Interactive Viewer

Compressed layered depth image bundle files can be loaded and interactively explored on the user's desktop machines in the interactive viewer application, the counterpart of the parallel rendering library in our framework.

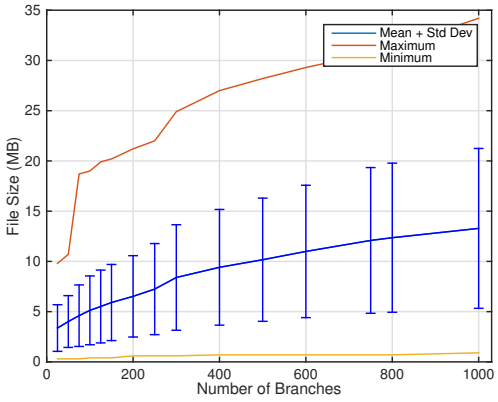


Figure 5: Impact of limiting the maximum number of branches in the automatic simplification of the *jet5* λ_2 data set on the compressed layered depth image file size, using the same camera perspective as in Figure 1. Depth image file size can vary greatly depending on the complexity of the scene captured in the rendering, however in general stays significantly smaller than the input data size of 134 MB.

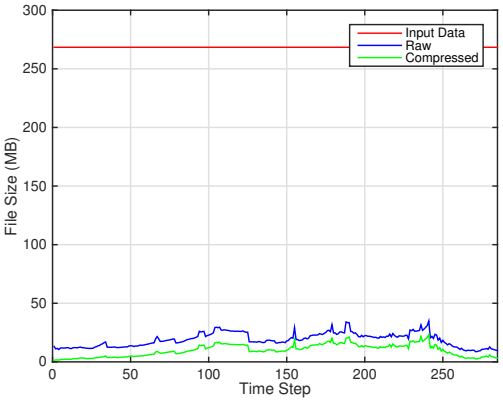


Figure 6: Comparison of raw and compressed layered depth image sizes in relation to the input data size of the *plate* λ_2 data set across the full time range, using 256 branches and the same camera perspective as in Figure 4. The reduction in file size is even more prominent for larger input data sets.

The interface is split in two parts: the visualization of the selected depth image and a tree widget used for the display and modification of the branch decomposition tree, including color, persistence, value range and volume of each branch. Users can manually or automatically select multiple branches by range or sorted minimum persistence/volume criterion as described in section 4. Additionally, picking branches in the visualization using the mouse cursor is an efficient way of selecting regions of interest. Picking can either be restricted to automatic selection of the front-most fragments or further guided by presenting all intersected segments in a cross-section interface. Selected branches are visually highlighted by distinct coloring. Given a set of selected branches, the user can apply filtering in order to simplify the visualization. Filters can even be combined by applying them on top of previous filtering steps. Undo is supported by storing the history of filtering operations.

Depth images are interactively updated and displayed in real time, being powered by hardware-accelerated rendering similar to the techniques outlined in section 5.2. Image data and branch properties as edited in the branch decomposition tree interface are transferred to shader storage buffer objects on the GPU and rendered to an offscreen framebuffer, which is afterwards displayed at flexible scale or exported to an image file. For each pixel, the fragments are back-to-front composited (they are already sorted by design), with shading being computed based on the stored fragment normal and branch coloring as defined for the respective branches. In addition to standard Phong illumination, we partially apply angle-based view-dependent transparency as presented in [HGH*10].

8. Results

We have tested our framework on two regular vector field data sets. The *jet5* data set (given on a $256\times512\times256$ grid over 3000 timesteps) results from a direct numerical simulation of a jet of high-velocity fluid entering a medium at rest and exhibits progressively finer vortical structures in the velocity field. Similarly, the *plate* data set (with a resolution of $1024\times256\times256$ over 285 timesteps) describes the mixing of fluid flowing past a plate at different speeds that undergo mixing due to viscous effects. Both data sets have been converted to regular scalar fields based on velocity, vorticity magnitude or the λ_2 vortex detection criterion [JH95]. Full examplary renderings of the data sets are depicted in Figures 1 and 4, respectively. An HSV color scale has been applied to all branches based on λ_2, where for each branch the average scalar branch value was approximated by the mean of saddle and extremum values stored with the branch.

All images depicted in this paper have been rendered at 1920x1080 resolution on a standard desktop workstation using an Intel Core i7-4770k quad-core CPU and a NVIDIA GeForce GTX 770 GPU. The complete topological segmentation of a single time step took less than one minute using the above hardware for both data sets. Rendering times varied depending on the chosen perspective, but typically ranged from 20 to 60 seconds. The final compression and storage of the resulting depth images was performed in less than one second.

8.1. Compression

The key goal of our contribution is to provide a flexible trade-off between storage memory consumption and interactive data exploration, essentially controllable by the branch decomposition simplification level used for layered depth image rendering.

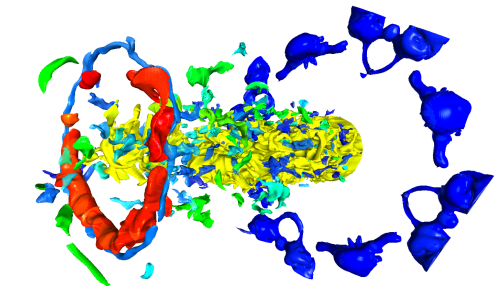
Figure 5 depicts the output bundle file sizes of our technique applied to the *jet5* data set in relation to several user-provided maximum branch numbers, using the same camera perspective as in Figure 1. For each number of branches, the graph shows the mean, standard deviation, minimum and maximum file sizes measured across the complete time range of the data set. Clearly, the mean file size is monotonously increasing with the number of branches. Notably, there is a small jump visible at 256 branches, when fragments are required to use shorts instead of bytes for branch index storage. However, in our studies, a maximum number of 256 branches has emerged as a very good compromise between file size and segmentation detail, providing a mean output size of approximately 7 MB, compared to the input data size of 134 MB for each scalar *jet5* time step, i.e., a size reduction of 95%.

The size reduction is even more prominent for full-view renderings of the larger *plate* data set as in Figure 4. Figure 6 illustrates the raw output size and compressed output size of our algorithm applied to each time step in relation to the constant input data size. In contrast to the great variation in file size due to the continuously increasing complexity of the *jet5* data set over time, the graph reflects the rather uniform complexity of the *plate* data set. Also, the graphs clearly highlight the fundamental reduction in data size compared to the input data even when storing the layered depth images in raw format, i.e., without additional zlib deflate compression, which usually achieves further compression ratios of about 50-60%.

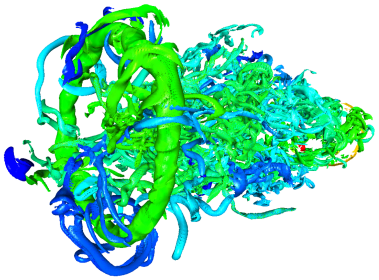
8.2. Analysis

By design, our approach features a powerful framework for hierachical simplification and automatic segmentation based on the branch decomposition of the contour tree, which is interactively controllable in the viewer application.

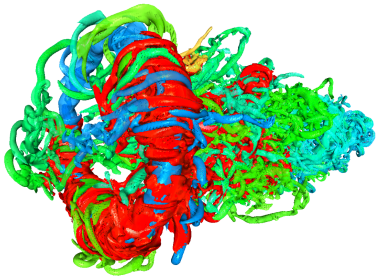
Figure 3a depicts a close-up on the central turbulences of the *plate* data set, which is located behind the main whirl visible in Figure 4. After an initial simplification of the whole data set to 256 branches prior to layered depth image rendering, the surrounding branches have been manually pruned in the viewer application. The remaining 90 branches visible in the depicted scene have been colored using a HSV color scale based on the branches' average λ_2 values. In the following Figures 3b, 3c and 3d, the branch decomposition subsequently has been further simplified to 50, 30 and 10 branches, respectively. Simplification was performed automatically based on sorted persistence as described in section 4, while maintaining the color scheme for non-discarded branches. One can clearly see the incremental reduction in complexity, which has been achieved with minimal user interaction and immediate visual feedback.



(a) velocity



(b) vorticity magnitude



(c) λ_2

Figure 7: Comparison of automatic persistence-guided simplification of the *jet5* data set into 256 branches and applying an HSV color scale based on the average branch value.

A comparison of applying automatic persistence-based simplification to the *jet5* data set is shown in Figure 7. In each image, the same scene is depicted consisting of 256 branches in total, with data having been constructed using either velocity (7a), vorticity magnitude (7b) or λ_2 (7c).

In our studies, the topological segmentation based on the branch decomposition of the contour tree has proven itself useful as a flexible representation of the major structures of interest occurring in the data sets, which furthermore provides an intuitive approach to simplification and filtering in both pre- and postprocessing.

9. Conclusion and Outlook

We have demonstrated the feasibility and potential of combining in situ topological analysis and compact image-based data representation. Our approach significantly reduces the amount of I/O bandwidth required to store the numerical results of high-fidelity numerical simulations running on large-scale parallel computer systems, while preserving flexibility in visualization.

Based on in situ contour tree analysis and simplification, we obtain a segmented representation of scalar fields contained in the simulation data at every time step. Together with the simplified contour tree, we store a rendering of this segmentation that describes all components visible in every pixel. Rendering can leverage hardware acceleration, as we have demonstrated by the GPU-based implementation of our rendering algorithm. The resulting compressed layered depth images can then be used in post-analysis to flexibly select specific subsets of the segmentation, and perform further topological simplification if required.

While our results already show substantial reductions in output file size, especially for larger data sets, our contribution is intended as a baseline demonstration investigating possible advantages of such an approach for the visualization of large-scale data. We anticipate that many enhancements and improvements of our approach are possible:

- Similar to Ahrens et al. [AJO*14], our technique could be easily extended to generate a complete in situ image data base from multiple perspectives, which can be combined in the viewer application to enable a flexible 3D data exploration, or even be used for reconstruction purposes.
- Multivariate topological methods such as *Joint Contour Nets* [CD14] might be investigated to obtain improved segmentations.
- A critical shortcoming of our current implementation is frame-to-frame temporal consistency. Since contour trees are computed and decomposed independently at each time step, the resulting contours can vary noticeably between time steps depending on the chosen automatic simplification criteria, potentially undermining analysis due to the lack of frame-to-frame coherence. This crucial problem could be addressed by incorporating feature tracking techniques into the branch selection and simplification process.
- The simplified contour tree stored with the compressed image files could be further annotated to compactly contain relevant properties of the original input data set, thereby improving the power and flexibility of the resulting visualization. However, not only the contour tree, but also the fragments can be used to compactly store local information of the intersected segment useful for later visualization.
- Notably, the general idea of our concept is not restricted to regular scalar data and is easily applicable to different kinds of potentially more complex data structures, providing means for topological segmentation and intersection.
- More sophisticated compression schemes might be used to further increase the compactness of the layered depth images generated.

We will investigate these possibilities in future work.

Acknowledgements

This project was funded in part under EU Career Integration Grant #304099.

References

[AJO*14] AHRENS J., JOURDAIN S., O'LEARY P., PATCHETT J., ROGERS D. H., PETERSEN M.: An image-based approach to extreme scale in situ visualization and analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Piscataway, NJ, USA, 2014), SC '14, IEEE Press, pp. 424–434. doi:10.1109/SC.2014.40. 2, 3, 9

[BEHP04] BREMER P.-T., EDELSBRUNNER H., HAMANN B., PASCUCCI V.: A topological hierarchy for functions on triangulated surfaces. *IEEE Transactions on Visualization and Computer Graphics 10* (2004), 2004. 2

[BPS97] BAJAJ C. L., PASCUCCI V., SCHIKORE D. R.: The contour spectrum. In *Proceedings of the 8th Conference on Visualization '97* (Los Alamitos, CA, USA, 1997), VIS '97, IEEE Computer Society Press, pp. 167–ff. 2

[BR63] BOYELL R. L., RUSTON H.: Hybrid techniques for real-time radar simulation. In *Proceedings of the November 12-14, 1963, Fall Joint Computer Conference* (New York, NY, USA, 1963), AFIPS '63 (Fall), ACM, pp. 445–458. doi:10.1145/1463822.1463869. 2

[BWM*12] BEKETAYEV K., WEBER G. H., MOROZOV D., ABZHANOV A., HAMANN B.: Geometry-preserving topological landscapes. In *Proceedings of the Workshop at SIGGRAPH Asia* (New York, NY, USA, 2012), WASA '12, ACM, pp. 155–160. doi:10.1145/2425296.2425324. 2

[BWP*10] BREMER P.-T., WEBER G., PASCUCCI V., DAY M., BELL J.: Analyzing and tracking burning structures in lean premixed hydrogen flames. *IEEE Transactions on Visualization and Computer Graphics 16*, 2 (2010), 1–1. doi:10.1109/TVCG.2009.69. 3

[BWT*11] BREMER P.-T., WEBER G., TIERNY J., PASCUCCI V., DAY M., BELL J.: Interactive exploration and analysis of large-scale simulations using topology-based data segmentation. *Visualization and Computer Graphics, IEEE Transactions on 17*, 9 (Sept 2011), 1307–1324. doi:10.1109/TVCG.2010.253. 3

[Car84] CARPENTER L.: The a -buffer, an antialiased hidden surface method. *SIGGRAPH Comput. Graph. 18*, 3 (Jan. 1984), 103–108. doi:10.1145/964965.808585. 3, 5

[CD14] CARR H., DUKE D.: Joint contour nets. *Visualization and Computer Graphics, IEEE Transactions on 20*, 8 (Aug 2014), 1100–1113. doi:10.1109/TVCG.2013.269. 9

[CPA*10] CHILDS H., PUGMIRE D., AHERN S., WHITLOCK B., HOWISON M., PRABHAT, WEBER G., BETHEL E.: Extreme scaling of production visualization software on diverse architectures. *Computer Graphics and Applications, IEEE 30*, 3 (May 2010), 22–31. doi:10.1109/MCG.2010.51. 1

[CS03] CARR H., SNOEYINK J.: Path seeds and flexible isosurfaces using topology for exploratory visualization. In *Proceedings of the Symposium on Data Visualisation 2003* (Aire-la-Ville, Switzerland, Switzerland, 2003), VISSYM '03, Eurographics Association, pp. 49–58. 2

[CSA00] CARR H., SNOEYINK J., AXEN U.: Computing contour trees in all dimensions. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms* (2000), SODA '00, pp. 918–926. 2, 4

[CSvdP04] CARR H., SNOEYINK J., VAN DE PANNE M.: Simplifying flexible isosurfaces using local geometric measures. In *Visualization, 2004. IEEE* (Oct 2004), pp. 497–504. doi:10.1109/VISUAL.2004.96. 2

[CSvdP10] CARR H., SNOEYINK J., VAN DE PANNE M.: Flexible isosurfaces: Simplifying and displaying scalar topology using the contour tree. *Comput. Geom. Theory Appl. 43*, 1 (Jan. 2010), 42–58. doi:10.1016/j.comgeo.2006.05.009. 3

[DBW*12] DEMIR D., BEKETAYEV K., WEBER G. H., BREMER P.-T., PASCUCCI V., HAMANN B.: Topology exploration with hierarchical landscapes. In *Proceedings of the Workshop at SIGGRAPH Asia* (New York, NY, USA, 2012), WASA '12, ACM, pp. 147–154. doi:10.1145/2425296.2425323. 2

[DN09] DORAISWAMY H., NATARAJAN V.: Efficient algorithms for computing reeb graphs. *Comput. Geom. Theory Appl. 42*, 6-7 (Aug. 2009), 606–616. doi:10.1016/j.comgeo.2008.12.003. 2

[EHNP03] EDELSBRUNNER H., HARER J., NATARAJAN V., PASCUCCI V.: Morse-smale complexes for piecewise linear 3-manifolds. In *Proc. of the Nineteenth Annual Symposium on Computational Geometry* (New York, NY, USA, 2003), SCG '03, ACM, pp. 361–370. doi:10.1145/777792.777846. 2

[FSME14] FREY S., SADLO F., MA K.-L., ERTL T.: Interactive progressive visualization with space-time error control. *Visualization and Computer Graphics, IEEE Transactions on 20*, 12 (Dec 2014), 2397–2406. doi:10.1109/TVCG.2014.2346319. 3

[FTAT00] FUJISHIRO I., TAKESHIMA Y., AZUMA T., TAKAHASHI S.: Volume data mining using 3d field topology analysis. *Computer Graphics and Applications, IEEE 20*, 5 (Sep 2000), 46–51. doi:10.1109/38.865879. 2

[GNP*05] GYULASSY A., NATARAJAN V., PASCUCCI V., BREMER P. T., HAMANN B.: Topology-based simplification for feature extraction from 3d scalar fields. In *Proceedings of IEEE Conference on Visualization* (2005). 2

[HGH*10] HUMMEL M., GARTH C., HAMANN B., HAGEN H., JOY K. I.: Iris: Illustrative rendering for integral surfaces. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (Nov. 2010), 1319–1328. doi:10.1109/TVCG.2010.173. 7

[JH95] JEONG J., HUSSAIN F.: On the identification of a vortex. *Journal of Fluid Mechanics 285* (1995), 69–94. 7

[LPG*14] LANDGE A. G., PASCUCCI V., GYULASSY A., BENNETT J. C., KOLLA H., CHEN J., BREMER P.-T.: In-situ feature extraction of large scale combustion simulations using segmented merge trees. In *Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Piscataway, NJ, USA, 2014), SC '14, pp. 1020–1031. doi:10.1109/SC.2014.88. 3

[MCTB12] MAULE M., COMBA J., TORCHELSEN R., BASTOS R.: Memory-efficient order-independent transparency with dynamic fragment buffer. In *Graphics, Patterns and Images (SIBGRAPI), 2012 25th SIBGRAPI Conference on* (Aug 2012), pp. 134–141. doi:10.1109/SIBGRAPI.2012.27. 5

[PCM04] PASCUCCI V., COLE-MCLAUGHLIN K.: Parallel computation of the topology of level sets. *Algorithmica 38*, 1 (2004), 249–268. doi:10.1007/s00453-003-1052-3. 2

[PCMS04] PASCUCCI V., COLE-MCLAUGHLIN K., SCORZELLI G.: Multi-resolution computation and presentation of contour trees. In *Proc. IASTED Conference on Visualization, Imaging, and Image Processing* (2004), pp. 452–290. 2, 4

[Pra10] PRANCKEVIČIUS A.: Compact normal storage for small g-buffers, Mar. 2010. URL: http://aras-p.info/texts/CompactNormalStorage.html. 6

[SA13] SEGAL M., AKELEY K.: Opengl 4.3 core profile specification, Feb. 2013. URL: https://www.opengl.org/registry/. 5

[TCM10] TIKHONOVA A., CORREA C., MA K.-L.: Visualization by proxy: A novel framework for deferred interaction with volume data. *Visualization and Computer Graphics, IEEE Transactions on 16*, 6 (Nov 2010), 1551–1559. doi:10.1109/TVCG.2010.215. 3

[TFT05] TAKAHASHI S., FUJISHIRO I., TAKESHIMA Y.: Interval volume decomposer: a topological approach to volume traversal, 2005. doi:10.1117/12.584257. 2

[TLB*11] THOMPSON D., LEVINE J., BENNETT J., BREMER P.-T., GYULASSY A., PASCUCCI V., PEBAY P.: Analysis of large-scale scalar data using hixels. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on* (Oct 2011), pp. 23–30. doi:10.1109/LDAV.2011.6092313. 3

[TTF04] TAKAHASHI S., TAKESHIMA Y., FUJISHIRO I.: Topological volume skeletonization and its application to transfer function design. *Graph. Models 66*, 1 (Jan. 2004), 24–49. doi:10.1016/j.gmod.2003.08.002. 2

[TTFN05] TAKESHIMA Y., TAKAHASHI S., FUJISHIRO I., NIELSON G. M.: Introducing topological attributes for objective-based visualization of simulated datasets. In *Proceedings of the Fourth Eurographics / IEEE VGTC Conference on Volume Graphics* (2005), VG'05, pp. 137–145. doi:10.2312/VG/VG05/137-145. 2

[vKvOB*97] VAN KREVELD M., VAN OOSTRUM R., BAJAJ C., PASCUCCI V., SCHIKORE D.: Contour trees and small seed sets for isosurface traversal. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry* (New York, NY, USA, 1997), SCG '97, ACM, pp. 212–220. doi:10.1145/262839.269238. 2

[WBP07] WEBER G., BREMER P.-T., PASCUCCI V.: Topological landscapes: A terrain metaphor for scientific data. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (Nov. 2007), 1416–1423. doi:10.1109/TVCG.2007.70601. 2

[WDC*07] WEBER G. H., DILLARD S. E., CARR H., PASCUCCI V., HAMANN B.: Topology-controlled volume rendering. *IEEE Transactions on Visualization and Computer Graphics 13*, 2 (Mar. 2007), 330–341. doi:10.1109/TVCG.2007.47. 3, 4

[WSH03] WEBER G. H., SCHEUERMANN G., HAMANN B.: Detecting critical regions in scalar fields. In *Proceedings of the Symposium on Data Visualisation 2003* (Aire-la-Ville, Switzerland, Switzerland, 2003), VISSYM '03, Eurographics Association, pp. 85–94. 2

[YHGT10] YANG J. C., HENSLEY J., GRÜN H., THIBIEROZ N.: Real-time concurrent linked list construction on the gpu. In *Proceedings of the 21st Eurographics Conference on Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2010), EGSR'10, Eurographics Association, pp. 1297–1304. doi:10.1111/j.1467-8659.2010.01725.x. 5