Journal of Physics:
Conference Series

PURPOSE-LED PUBLISHING™

OPEN ACCESS

Visualization of large scale time-varying scientific data

To cite this article: Han-Wei Shen 2006 *J. Phys.: Conf. Ser.* **46** 535

View the article online for updates and enhancements.

You may also like

- Challenges in Scientific Data Communication from Low-mass Interstellar Probes
David G. Messerschmitt, Philip Lubin and Ian Morrison
- Observation data pre-processing and scientific data products generation of POLAR
Zheng-Heng Li, Jian-Chao Sun, Li-Ming Song et al.
- An adaptive continual learning framework for dynamic condition monitoring of reciprocating compressors
Huixin Tian, Yuhang Kang, Zijun Li et al.

This content was downloaded from IP address 173.177.195.252 on 26/02/2026 at 16:41

Institute of Physics Publishing Journal of Physics: Conference Series **46** (2006) 535–544
doi:10.1088/1742-6596/46/1/074 SciDAC 2006

Visualization of large scale time-varying scientific data

Han-Wei Shen

Department of Computer Science and Engineering
The Ohio State University
395 Dreese Lab, 2015 Nail Avenue, Columbus, OH 43210

E-mail: hwshen@cse.ohio-state.edu

Abstract.

New challenges for scientists have emerged in the past several years as the size of data generated from simulations has experienced an exponential growth. One major factor that is contributing to the growth of data size is the increasingly widespread ability to perform very large scale time-varying simulations. To analyze complex dynamic phenomena from a time-varying data set, it is necessary to navigate and browse the data in both the spatial and temporal domains, select data at different resolutions, experiment with different visualization parameters, and compute and animate selected features over a period of time. In this paper, we present several algorithms for visualizing large scale time-varying scientific data including: (1) Lossless spatio-temporal data encoding and indexing schemes allowing for interactive visualization of time-varying data at arbitrary spatial and temporal scales. (2) Coherence based accelerated visulaization algorithms (3) Time-varying feature enhancement and tracking algorithms. Our goal is to minimize visualization computation cost, to minimize data transfer (network transmit and disk I/O) cost, to maximize the user's ability to interrogate time-varying data in different spatial and temporal scales, and to detect and track important time-varying features.

1. Introduction

The rapid advance of computer hardware and software has made it possible for scientists to significantly increase the scale and accuracy of their simulation models to solve previously unsolvable problems. Various computational science and engineering applications share a common theme in that they all produce a large amount of numerical output. To glean insights into the data, *Scientific Visualization* has become a key enabling technology for scientific discovery through advanced computing. The efficiency of scientific visualization lies primarily on its ability in presenting information concisely, and allowing the user to have direct interaction with the data.

New challenges for scientific visualization research have emerged in the past several years as the size of data generated from simulations experienced an exponential growth. The average size of data generated from a large-scale computational simulations has grown from hundreds of megabytes to the current tera- to peta-bytes scales. Not surprisingly, data of this scale is overwhelming and thus makes the tasks of interactive visualization very difficult.

A major factor that is contributing to the exponential growth in the size of data is scientists' ability to perform large scale time-varying simulations. To analyze complex dynamic phenomena

© 2006 IOP Publishing Ltd

from a time-varying data set, it is necessary to navigate and browse the data in both the spatial and temporal domains, select data at different resolutions, experiment with different visualization parameters, and compute and animate selected features over a period of time. To facilitate exploratory visual data analysis, it is very important that the visualization software can compute, animate, and track desired features at an interactive speed. However, the nature of time-varying data sets makes run-time navigation very challenging since the time to load and transform the raw data into various visual forms can be very long.

Although intensive research has been focused on optimizing the performance of large scale data visualization [9], most of the existing methods were not designed specifically for time-varying data. If one tries to use the conventional algorithms designed for static data to visualize time-varying data, the performance often suffers. To date, several challenges still remain when considering an end-to-end processing of time-varying data. First of all, the existing visualization systems do not have explicit data management schemes optimized for time-varying data. Secondly, since general data representations and processing strategies are lacking, integrating different techniques into a general visualization environment is difficult. Furthermore, effective visualization techniques that can extract, display and track complex time-dependent features are still lacking. There is a great need to develop comprehensive strategies and frameworks to facilitate an efficient processing of large-scale time-varying data. Algorithms are needed to minimize visualization computation cost, to minimize data transfer (network transmit and disk I/O) cost, to maximize the user's ability to interrogate time-varying data in different spatial and temporal scales, and to detect and track important time-varying features.

In the remainder of this paper we will first present a multresolution data management framework for large scale time-varying data. We then discuss how temporal coherence can be utilized to speed up time-varying visualization computation. To enhance and compare time-varying features, a high dimensional projection technique using volume rendering is presented. Finally, to allow the scientist to isolate local features and track their evolution, we present a isocontour and interval volume tracking algorithm based on higher dimensional geometry.

2. Spatio-Temporal Volume Data Encoding and Management

Having the ability to visualize data at arbitrary spatial and temporal resolutions allows the user to identify features at different scales, and balance visualization quality and computation speed. Although data encoding for spatial volumes and videos has been studied intensively, transformation and management of large scale three-dimensional time-varying volumes require special treatment. To address this need, we develop an effective data encoding and management scheme, called *wavelet-based time-space partitioning (WTSP) tree*, to facilitate interactive browsing of spatio-temporal multiresolution data. In addition to data compression, the primary goal of our work is to de-correlate the time-varying data into a range of spatial and temporal levels of detail for the purpose of enabling rapid run-time data retrieval, reconstruction, and rendering.

The WTSP tree [19] is a space-time hierarchical data structure used to organize multiresolution time-varying volume data. The underlying data transformation scheme used to create the multiresolution hierarchy is a blockwise two-stage wavelet transform and compression process, as illustrated in Figure 1. The first stage is to build a spatial hierarchy in the form of an octree (similar to the *wavelet tree* [5]) for each time step, where each node in the tree represents a subvolume with a certain spatial resolution at a particular time step. As shown in Figure 1 (a), the volumetric data for one time step is subdivided into a sequence of data blocks/subvolumes of the same size (assuming each has n voxels). We perform a 3D wavelet transform for each of the volume blocks. This will produce a low-pass filtered subblock of size $n/8$ and wavelet coefficients of size $7n/8$. The wavelet coefficients are compared against a user-specified threshold and clamped to zero if they are smaller than the threshold. The low-pass filtered subblocks from

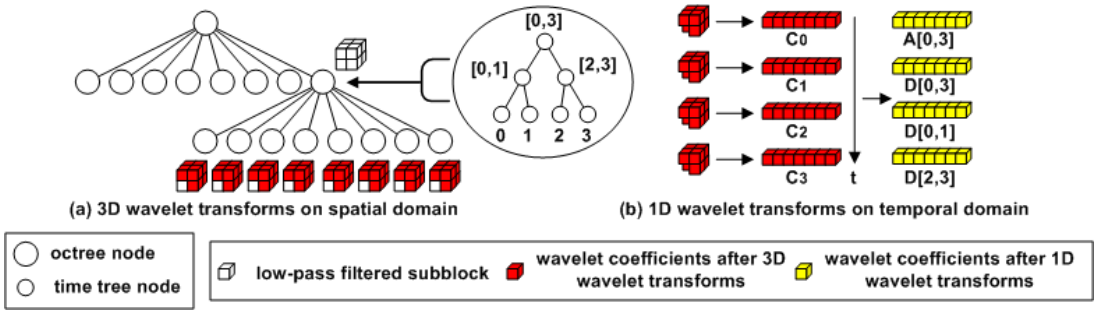
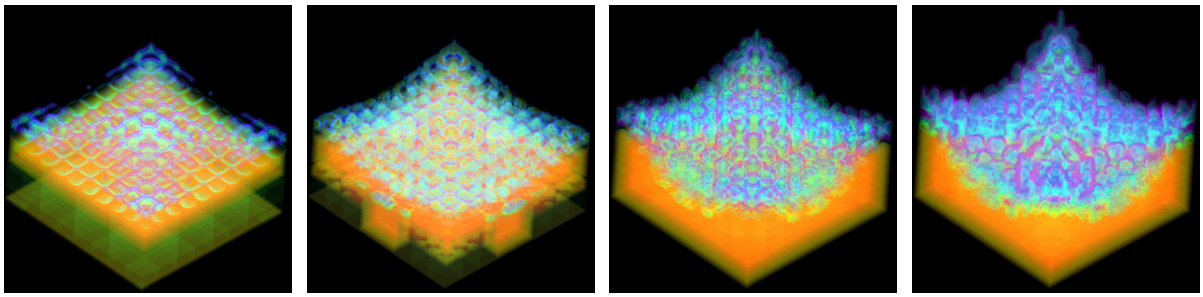


Figure 1. The conceptual diagram of constructing the WTSP tree. In this figure, the time-varying data has four time steps. Ct are the wavelet coefficients at time step t. $A[t_1, t_2]$ and $D[t_1, t_2]$ represent the averages and differences of the coefficients across time span $[t_1, t_2]$ respectively. During the construction, we compute and store $A[0, 3]$, $D[0, 3]$, $D[0, 1]$ and $D[2, 3]$. At run time, they are used to reconstruct $A[0, 1]$, or $A[2, 3]$, or one of Ct $(0 \leq t \leq 3)$ by traversing the time tree from top down, depending on the time step in query and the error tolerances.

eight neighboring subvolumes are collected and grouped into a single block of n voxels, which becomes the parent node of the eight subvolumes in the spatial octree hierarchy (note that the wavelet coefficients are kept at the child nodes, while the low-pass filtered subblocks are passed up to the parent). We recursively apply this 3D wavelet transform and subblock grouping process until the root of the octree is reached, where a single block of size n is used to represent the entire volume. This process is repeated to construct one multiresolution spatial wavelet tree for every time step.

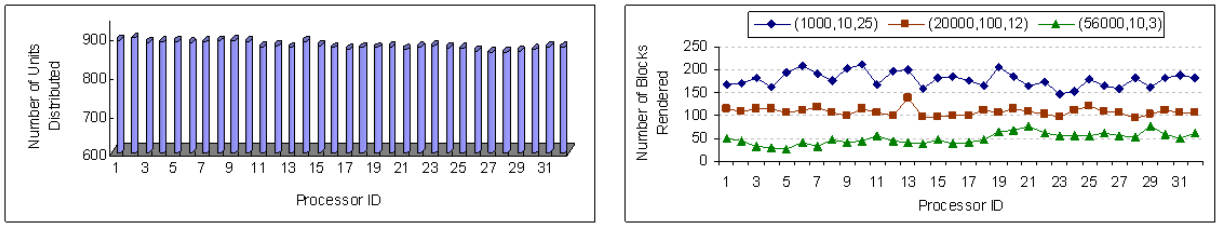
In the second stage, to create the temporal hierarchy from the octrees of all time steps, we apply 1D wavelet transforms to the wavelet coefficients associated with octree nodes having the same spatial location and resolution across the time sequence. Using Haar wavelets, this will produce a binary time tree similar to the *error tree* algorithm [10], as illustrated in Figure 1 (b). The wavelet coefficients resulting from 1D wavelet transforms are then compressed using run-length encoding combined with a fixed Huffman encoder [5]. This bit-level run-length encoding scheme exhibits good compression ratio if many consecutive zero subsequences are present in the wavelet coefficient sequence, and is very fast to decompress. For the root nodes of the octrees, the temporal hierarchy is built from the low resolution data blocks rather than wavelet coefficients. As a result, the 1D wavelet transform process merges all the spatial octrees across time into a single unified spatio-temporal hierarchical data structure. In essence, the WTSP tree is an octree (spatial hierarchy) of binary trees (temporal hierarchy). There is only one octree skeleton, and at each octree node, there is a binary time tree. Each time tree spans the entire time sequence and combines data from multiple octrees.

With an additional partitioning and distribution scheme, as described in [19], WTSP trees allow us to render large scale time-varying data at desired spatial and temporal resolutions using PC clusters. Figure 2 shows the rendering of a Richtmyer-Meshkov Instability (RMI) turbulence simulation dataset generated at Lawrence Livermore National Laboratory at selected time steps. We were able to perform a lossless compression of the data from 30 GB (1024x1024x960x30) to about 5.8 GB using wavelet transform with a lifting scheme. At run time, the user can choose different spatial and temporal error thresholds (denoted as se and te in Figure 2 to render the data at different quality. When the spatial and temporal error tolerances were higher, data blocks of lower resolutions were reconstructed, which resulted in a smaller number of blocks being rendered and thus faster speed. It can be observed that, finer details of the data are kept when reducing the error tolerances, but images of reasonable quality can still be obtained at lower



(a) $(20000, 10, 1)$ (b) $(55000, 50, 8)$ (c) $(52410, 20, 15)$ (d) $(52200, 30, 32)$

Figure 2. Rendering of the RMI data set for selected time steps. All images were rendered with the same viewing parameters. A total of 536, 743, 1317 and 1625 blocks were rendered for $(se, te, t) = (20000, 10, 1)$, $(55000, 50, 8)$, $(52410, 20, 15)$ and $(52200, 30, 32)$ respectively.



(a) (b)

Figure 3. (a) The number of distribution units distributed to each of the 32 processors. (b) The number of data blocks rendered at each of the 32 processors with three different queries of (se, te, t) for the RMI data set. A total of 1597, 3462, and 5723 blocks were rendered for $(56000, 10, 3)$, $(20000, 100, 12)$ and $(1000, 10, 25)$ respectively.

resolutions. In terms of parallel speedup, We were able to achieve about 95% of CPU utilization when using 32 processors. Figure 3 (a) shows the number of distribution units distributed to each of the 32 processors, and (b) shows the number of data blocks rendered at each of the 32 processors, when three different time steps and tolerances of spatial and temporal errors were used. A distribution consists of a group of data blocks of various spatial and temporal errors. It can be seen that good load-balancing was achieved, because the processors rendered approximately equal numbers of blocks.

3. Accelerated Time-Varying Visualization Algorithms

One strategy for accelerating time-varying visualization algorithms is to utilize the data coherence. In general, two types of coherence can be usually observed in a time-varying volume dataset. One is the spatial coherence, which refers to the fact that voxels in local regions tend to have values that are very close to each other. The other is the temporal coherence, which exists in regions where the underlying data do not change much over time. Exploiting strong similarities between data in consecutive time steps can avoid redundant visualization computation and data transfer. In addition, temporal coherence can be used to guide the focus of explorations, as well as the allocation of computational resources, so that regions with different temporal complexities can be visualized at different resolutions. In the following, we present two scalar field visualization algorithms that utilize temporal coherence.

3.1. Time-Varying Isosurface Extraction

An isosurface represents regions that have a constant value in a three-dimensional scalar field. Displaying isosurfaces is a useful technique for analyzing scalar data due to its effectiveness in revealing the spatial structures of the field's value distribution. To compute the isosurface, Lorensen and Cline [8] proposed a Marching Cubes algorithm which extracts small polygon patches from individual cells in the field. The Marching Cubes algorithm is simple and robust. However, its process of linear search for isosurface cells is expensive. To improve the performance, previously we have conducted research on reducing the cost of finding isosurface cells [15, 7].

The main focus of this work on time-varying isosurface extraction is to devise a new search index structure in a time-varying field so that the storage overhead is kept small, while the performance of the isosurface extraction remains high. In addition, the new search index structure should allow a flexible control of the tradeoff between performance and storage space and, thus, can be used in different computing environments.

In general, isosurface cells are located by inspecting each cell's extreme values (minimum and maximum values). If the cell's minimum value is smaller than the isovalue, and the maximum value is greater than the isovalue, this cell is intersected by the isosurface. To speed up the search process, extreme values of the cells at every time step are sorted into a search index structure. While the use of the search index can achieve an optimal performance in extracting the isosurface, the storage overhead is usually quite high for a time-varying field. To reduce the storage overhead while still maintain the high performance in isosurface extraction, we characterize each cell in the field based on its extreme values and the variation of the extreme values over time. Consider a cell that has a high temporal coherence and, thus, a small scalar variation in time. Such a cell, in a period of several time steps, may be referenced by that cell's overall extreme values in the time span. On the other hand, for a cell that has little coherence and, thus, a high scalar variation, the cell is referenced by separate extreme values at individual time steps. Cells are stored into a binary tree data structure called *Temporal Hierarchical Index Tree* [13] based on their temporal variations. Cells that have a small amount of variation over time are placed into the nodes that are closer to the root where each node covers several time steps. Cells with a larger variation are placed in multiple nodes of the tree close to the leaves, each for a short time span. When generating an isosurface, a simple traversal will retrieve the set of nodes that contains all the candidate isosurface cells at a given time step.

We have applied our algorithm to several large-scale time-varying datasets. The test results have shown that we can achieve disk space savings to as much as 80% for creating the isosurface cell search structure. Even for data sets that have higher scalar variation we were still able to achieve about $60 - 70\%$ of saving in storage. Although the use of aggregated min/max value across several time steps to search isosurface cells would generate some false positives, we have found that the performance overhead was about 15% for the data that we received an 80% of storage saving. The false positive cells won't affect the correctness of the isosurfaces because they will be eliminated at the triangulation step. The two images on the left of Figure 4 show examples of time-varying isosurfaces generated from a supernova simulation (TSI) dataset.

3.2. Time-Varying Volume Rendering

Direct volume rendering has been widely used in various areas such as medical imaging, structure analysis, earth science, and computational fluid dynamics. The advantage of using direct volume rendering techniques is that both opaque and translucent structures can be visualized at the same time. Unfortunately, the computation cost of direct volume rendering is often too high for large data sets. To speed up the visualization process, we propose to decouple the temporal and spatial domains for a better utilization of both the temporal and spatial coherence. We devised a hierarchical data structure, called Time-Space Partitioning (TSP) [14] tree, that can capture both the temporal and the spatial coherence in a time-varying field.

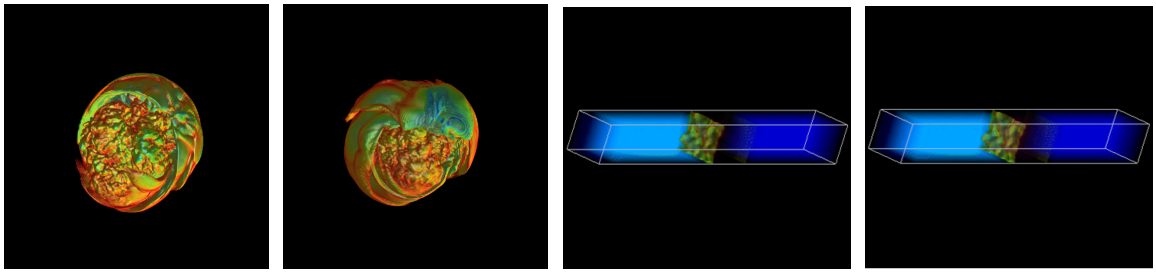


Figure 4. Time-Varying isosurfaces and volume rendering generated from the the TSI entropy and a showckwave data sets

The TSP tree is a time-supplemented octree. The skeleton of a TSP tree is a standard complete octree, which recursively subdivides the volume spatially until all subvolumes reach a predefined minimum size. To store the temporal information, each TSP tree node itself is a binary tree. The binary tree bisects the time span $[0, t]$ associated with the time-varying field until a unit time step is reached. Figure 5 depicts the TSP tree and one of its tree nodes in the form of a binary time tree. A quadtree is used for illustration purpose.

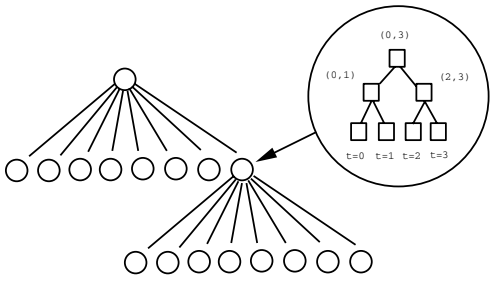


Figure 5. The TSP tree's skeleton is an octree, and each of the TSP tree nodes is a binary time tree. In the example here, the time-varying field has four time steps.

Every node in the binary time tree associated with a TSP tree node represents the same subvolume in the spatial domain but a different time span. The information stored in a binary time tree node includes the time-averaged low resolution data, a measurement of the subvolume's spatial error in the given time span, and a measurement of the subvolume's temporal error in the given time span. For a time-varying volume series, the TSP tree only needs to be constructed once and can then be employed repeatedly. To perform volume rendering at run time, the TSP tree is first traversed to identify the subvolumes that satisfy the user-supplied spatial and temporal error tolerances. Our tree traversal algorithm consists of traversing the TSP tree's octree skeleton and traversing the binary time tree associated with each encountered TSP tree node. The tolerance for the spatial error provides a stopping criterion for the octree traversal so that regions having tolerable spatial variations are rendered using lower resolution data. The tolerance for the temporal error is used to identify regions where the rendering results can be reused for multiple time steps due to their small temporal variations. The located subvolumes are then rendered in the correct order to construct the final image.

The main advantage of using TSP trees for rendering time-varying volume data is that regions with high temporal coherence only need to be rendered once for the entire coherent time span, which can reduce the amount of I/O and rendering that are required. The two images on the right of Figure 4 show an example of rendering a shockwave dataset. The data were generated from a simulation of the unsteady interaction of a planar shock wave with a randomly-perturbed

contact discontinuity [11]. For the first time step of the volume, a complete rendering was needed, so no speedup was gained. However, for the subsequent time steps, the temporal coherence was utilized, and only a portion of the volume bricks at each time step needed to be re-rendered. In our tests, we were able to reduce the I/O amount to about 13% of the original data, while speed up the rendering up to eleven times without degrading the rendering quality.

4. High-Dimensional Projections

Currently, two methods of visualization are commonly used for time-varying data. One is to generate visualization from certain time steps of interest and display those images individually. In this manner, the continuity and connection between time steps is lost. The information of a single time step is present, but the context of the surrounding earlier and later time steps is not immediately available. The second method is to create an animation from the data set. While the context of surrounding time steps is provided, it relies on the viewer's memory and cognitive abilities to tie together the features' spatio-temporal relationships. In this section, we present a new rendering technique to provide the user with better insight to the data set's temporal characteristics.

For a time-varying data set, rather than considering space and time as separate components, alternatively the data can be treated as a four-dimensional data field. Already in the nineteenth century, mathematicians had fashioned models to show the sequence of hypercube slices in various directions [1]. Artists, photographers, and the early pioneers of cinematography explored the co-mingling of space and time, such as Marcel Duchamp's painting, *Nude Descending a Staircase (No. 2)*, and Etienne-Jules Marey's chronophotographic images [3], to better illustrate the motion and flow of the figures moving over time.

To allow the user to compare features over time, we use high dimensional projections to assist understanding of spatio-temporal characteristics in the data sets. To perform projection in four dimensions, we approach the concept by specification of hyperslices in 4D space and projection of these slices to 2D images. The concept of slicing 3D volumes can be readily extended to 4D volumes. One simple example of hyperslices is a 3D volume coincides with the spatial domain that defines the volume grid. For more complex hyperslices, it is less intuitive to specify the location and orientation of a hyperplane in 4D volumes. Given that, we devise a more intuitive method for interactive specification of arbitrary hyperslices [20, 21]. With the hyperslices, we can go one step further to perform projection of the hyperplanes by extending parallel projection in 3D to perform 4D projection. To render a 3D volume , what is typically done is to move a 2D image plane along its normal direction subsampling and integrating the underlying 3D volume. Now in the four dimensional case, we can sample the 4D volume using a sequence of hyperslices that are parallel to each other along the normal direction of the hyperplanes. The result of the 4D projection is a 3D volume where each voxel in the resulting volume is an integration of space-time samples. To visualize this hyperprojection volume, standard volume rendering techniques can be used to project the volume interactively down to a 2D image plane.

A unique issue to our 4D projection approach is the design of appropriate space-time integration functions for highlighting and tracking temporal features. Imagine a ray is being cast from each voxel along the time dimension to perform a temporal integration. The transfer functions and the integration schemes in use will determine the usefulness of the integration result. We have been experimenting with different integration functions and received promising results [20, 21]. On the left of Figure 6, isocontours were rendered using an additive composition technique, where we take a summation of the colors for data from all time steps at the same position, and do a normalization step at the end. Blue indicates the oldest time step, while red indicates the most recent time step. Green is a time step that is the median between the newest and the oldest time step. The opacity of the voxel at a single time step before being integrated into the result value is a modulation of time opacity and data opacity. In the example, we can

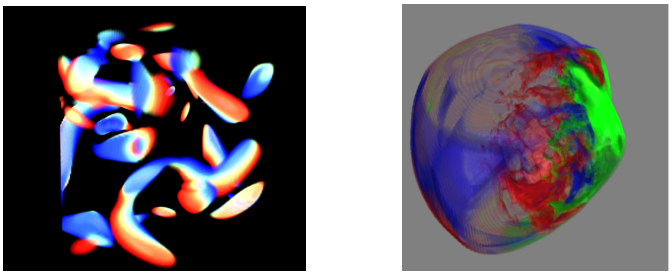


Figure 6. Different temporal integration functions, right: additive, left:minimum intensity

see several isocontours as they progress over time and space; When features at all time steps overlap in space, they appear as white, or possibly appear as cyan, magenta, or yellow, if only a few time steps occupy the same space. This additive scheme allows us to see whether features at different time steps overlap, and by how much they overlap. On the right of Figure 6 we show a minimum intensity technique, which colors each voxel during the temporal integration by the time step that has the minimum value. Visualizing the min/max values throughout the entire time sequence for every voxel in the volume in a single picture allows us to pick out the "hot spot" in a time sequence, i.e., the user can see that in a region which time step had the extreme value.

5. Tracking Time-Dependent Isocontour and Interval Volumes

Isocontours and interval volumes are commonly used for characterizing features that can be described by the underlying scalar data values. An isocontour represents a level set of constant value, while an interval volume represents points whose values lying within a certain closed interval [4, 12]. When analyzing a time-dependent data set, a straightforward animation of isocontours or interval volumes in time may not be always effective, since the visualization can contain a large number of small components, and each of which may experience complex evolution over time (Figure 7). To study the temporal characteristics of spatio-temporal phenomena from a long time sequence, having the ability to isolate specific local features and track their evolutions in time can be more visually effective and computationally efficient.

We have developed algorithms to track time-dependent isocontours and interval volumes from user-selected local components. When a scientist is presented with rendering of an isosurface or interval volume at a particular time step, he or she can select a connected component of interest, and then follows its evolution in time. From the selected component, topological changes of the feature over time, such as *dissipation*, *creation*, *bifurcation*, *amalgamation*, and *continuation*, as well as the time steps when those key events occur will be automatically identified and presented to the user. Previously, Silver and Wang [16, 17, 18] proposed volume tracking algorithms to follow features from time-varying data. In their method, features manifested as interval volumes are identified and stored in octrees and link list data structures. Then, correspondences between features in consecutive time steps are established. While effective, in their algorithms the global feature computation and matching can be computationally expensive and incur storage overhead. We propose an alternative approach which can track time-varying local features and identify their evolutionary events more efficiently.

We use a similar criterion of spatial overlapping, as in Silver and Wang's work, to define feature correspondence. We are currently working on relaxing this constrain to allow tracking of fast moving objects that do not overlap. The spatial overlapping assumption allows us to track the evolution of an isocontour in a d dimensional time-varying data set by first computing the isocontour in $d+1$ dimensions and then slice the resulting hyper-isocontour along the time

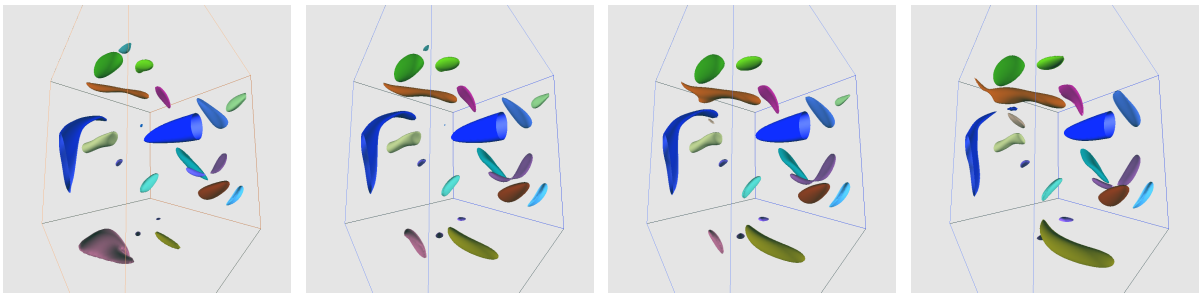


Figure 7. Time varying features from the vorticity data set are tracked and the tracking result determines the coloring of the features. Time progresses from left to right.

axis to get the feature at the desired time step. Compared to the previous work by Silver and Wang, the main benefit of our approach is to allow local propagation and tracking of isocontours and interval volumes. In addition, important topological events and critical time steps can be detected by inspecting the high dimensional geometry. We use the isocontouring algorithms proposed by Bhaniramka *et al* [2] to compute the four dimensional isocontours. To track 3D interval volumes, similarly, we first extract and then slice a four dimensional interval volume. Previously, researchers have proposed algorithms to extract interval volumes in three dimensions [4, 12]. However, there is no algorithm for computing interval volumes in four dimensions. We have designed an algorithm to compute four dimensional interval volumes using five dimensional isocontours. More detail can be found in [6].

With the high dimensional geometry, it is possible to detect critical time steps automatically at which the features of interest undergo changes in their topological configurations. This can be done by analyzing the 4D geometry (isocontours or interval volumes). A 4D isocontour component is a tetrahedral mesh embedded in four dimensional space. Each vertex of the tetrahedral mesh has coordinates (x,y,z,t). To compute the 3D isocontour component at a particular time step τ, we can slice the 4D tetrahedra using the t coordinates at the tetrahedra vertices. This is equivalent to treating the 4D mesh as a normal 3D mesh, with the time values as the scalar values defined over the tetrahedron vertices. In this case, animating an isocontour component over time is equivalent to computing the isocontours from the 3D mesh by sweeping the isovalue τ. The critical points of this mesh, i.e., local minima, local maxima, and saddle points at the vertices, indicate when and where the topology of the isocontour will change if the isocontour is animated across those time steps. A local minimum corresponds to *creation* of a 3D component while a local maximum corresponds to *dissipation* of a 3D component. Saddle points may correspond to either *amalgamation* or *bifurcation* of 3D components. *Continuation* occurs at regular vertices of the mesh. The time values associated with the critical points indicate when those topological events happen. Figure 7 show an example of our tracking results.

6. Conclusions

This paper presents several algorithms for visualizing large scale time-varying scientific data. Considering an end-to-end visualization process, the algorithms described in the paper represents several key steps in the visualization pipeline to facilitate efficient and effective visualization of complex time-varying data. The multiresolution spatio-temporal hierarchy allows the users to visualize features at different scales, and direct computation resources to more important regions and time spans. The coherence based visualization algorithms can minimize the amount of I/O and computation that is required so a quick browse of data becomes possible. High dimensional projection and isocontour tracking algorithms allow the user to compare and follow the features

in time to discover the temporal trends of the underlying data. Future work will be focused on the integration of the proposed techniques and also the deployment of the algorithms in large scale parallel computing environments to handle the emerging peta-scale applications.

7. Acknowledgments

This work was supported by DOE Early Career Principal Investigator Award DE-FG02-03ER25572, NSF ITR grant ACI-0325934, and NSF Career Award CCF-0346883. We thank Mark Duchaineau, Tony Mezzacappa, and Ravi Samtaney for the datasets.

References

[1] T.F. Banchoff. *Beyond the Third Dimension: Geometry, Computer Graphics, and Higher Dimensions*. Scientific American Library, 1990.

[2] P. Bhaniramka, R. Wenger, and R. Crawfis. Isosurfacing in higher dimensions. In *Proceedings of Visualization '00*, pages 267–273. IEEE Computer Society Press, Los Alamitos, CA, 2000.

[3] M. Braun. *Picturing Time: The Work of Etienne-Jules Marey*. The University of Chicago Press, 1992.

[4] I. Fujishiro, Y. Maeda, H. Sato, and Y. Takeshima. Volumetric data exploration using interval volume. *IEEE Transactions on Visualization and Computer Graphics*, 2(2), 1996.

[5] S. Guthe, M. Wand, J. Gonser, and W. Straber. Interactive rendering of large volume data sets. In *Proceedings of Visualization '02*, pages 53–60. IEEE Computer Society Press, Los Alamitos, CA, 2002.

[6] G. Ji, H.-W. Shen, and R. Wenger. Volume tracking using higher dimensional isocontouring. In *Proceedings of Visualization '03*. IEEE Computer Society Press, Los Alamitos, CA, 2003.

[7] Y. Livnat, H.-W. Shen, and C.R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.

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

[9] K.-L. Ma. Large scale data visualization. *IEEE Computer Graphics and Applications*, 21(4), 2001.

[10] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-Based Histograms for Selectivity Estimation. In *ACM SIGMOD '98*, pages 448–459, 1998.

[11] D.I. Meiron and R. Samtaney. 3D simulations of the Richtmyer-Meshkov instability with re-shock. *Bulletin of the American Physical Society*, 43(9):2104, 1998.

[12] G. Nielson and J Sung. Interval volume tetrahedralization. In *Proceedings of Visualization '97*. IEEE Computer Society Press, Los Alamitos, CA, 1997.

[13] H.-W. Shen. Isosurface extraction in time-varying fields using a temporal hierarchical index tree. In *Proceedings of Visualization '98*, pages 159–166. IEEE Computer Society Press, Los Alamitos, CA, 1998.

[14] H.-W. Shen, L.J. Chiang, and K.L. Ma. A fast volume rendering algorithm for time-varying field using a time-space partitioning (TSP) tree. In *Proceedings of Visualization '99*. IEEE Computer Society Press, Los Alamitos, CA, 1999.

[15] H.-W. Shen, C.D. Hansen, Y. Livnat, and C.R. Johnson. Isosurfacing in span space with utmost efficiency (ISSUE). In *Proceedings of Visualization '96*, pages 287–294. IEEE Computer Society Press, Los Alamitos, CA, 1996.

[16] D. Silver and X. Wang. Volume tracking. In *Proceedings of Visualization '96*, pages 157–164. IEEE Computer Society Press, Los Alamitos, CA, 1996.

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

[18] D. Silver and X. Wang. Tracking scalar features in unstructured data sets. In *Proceedings of Visualization '1998*, pages 79–86. IEEE Computer Society Press, Los Alamitos, CA, 1998.

[19] C. Wang, J. Gao, L. Li, and H. Shen. A multiresolution volume rendering framework for large-scale time-varying data visualization. In *Proceedings of Volume Graphics'05*, pages 11–17. IEEE Computer Society Press, Los Alamitos, CA, 2005.

[20] J. Woodring and H.-W. Shen. Chronovolumes: A direct rendering technique for visualizing time-varying data. In *Proceedings of Volume Graphics Workshop 2003*. Eurographics Association, 2003.

[21] J. Woodring, C. Wang, and H.-W. Shen. High dimensional direct rendering of time-varying volumetric data. In *Proceedings of Visualization '03*. IEEE Computer Society Press, Los Alamitos, CA, 2003.