Real Time Feature Extraction and Tracking in a Computational Steering Environment

J. Chen, D. Silver and M. Parashar
Dept. of Electrical and Computer Engineering and CAIP Center,
Rutgers University,
P.O. Box 909, Piscataway, NJ 08855-0909
jianc, silver,parashar@caip.rutgers.edu

Keywords: Visualization, Feature extraction, Tracking, Computational steering, Distributed algorithm

ABSTRACT

Large distributed time-varying simulations are common in many scientific domains to study the evolution of various phenomena. These simulations produce thousands of timesteps which must be analyzed and interpreted. For datasets with evolving features, feature analysis and visualization tools are crucial to help interpret all the information. For example, it is usually important to know how many regions are evolving, what are their lifetimes, do they merge with others, how does the volume/mass change, etc. To be effective these visualization and analysis routines must also be parallelized in order to operate on the data where that data resides. Furthermore, interacting with the routines as the simulations are ongoing can aid in the analysis. In our previous work, we have developed a methodology for analyzing time-varying datasets which tracks 3D amorphous features as they evolve in time. In this paper, we describe the full parallel feature extraction and tracking algorithm within a *computational steering* environment for parallel and distributed simulations. We demonstrate how one can interact with the code and show various examples within ongoing computations.

1. INTRODUCTION

Time varying simulations are common in many scientific domains to study the evolution of phenomena or features. The data produced in these simulations is massive. Instead of just one dataset of 512^3 or 1024^3 (for regular gridded simulations) there could now be hundreds to thousands of timesteps. For datasets with evolving features, feature analysis and visualization tools are crucial to help interpret all the information and highlight the underlying physical processes [21][12][18]**.** For example, it is usually important to know how many regions are evolving, whether they merge with other regions, and how their volume may change over time. Therefore, feature based approaches, such as feature tracking and feature quantification are needed to follow identified regions over time. The first step is to define what regions are of interest, the second is to track them in all of the timesteps. Another important application of feature based approach is in data mining. These features can be catalogued to build a scientific database. This will help the scientist to relate the phenomenon of interest with previous simulations and perform event matching. A basic framework for analyzing time-varying datasets was presented in [21][22] and an overview is shown in Figure 1**.** The goal of the process is to obtain dramatic data reduction and thus help scientists quickly focus on a few features or events of interest.

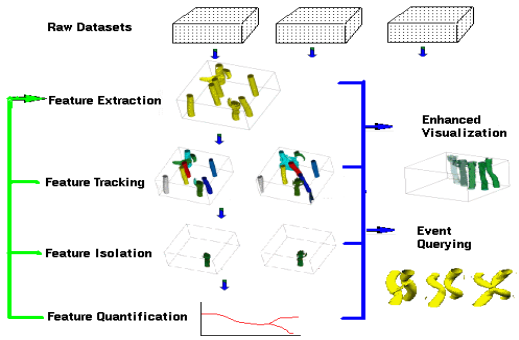

Figure 1. A process for visualizing 3D continuum datasets

As can be seen in Figure 1, the first step is to identify the features of interest and to define them. In [21][22][19] features are defined as thresholded connected components. The features are extracted from each dataset and then correlated over time using the algorithm in [21]. Once all the features are correlated, they can be quantified, compared, measured, etc. to obtain a full analysis of the evolution of each feature. This information constitutes "meta-data" which can be used for general querying. One example is *event querying*, i.e. retrieve all timesteps that demonstrate a particular event like large object merging. A database can eventually be built allowing the scientist understand the simulation in the context of all the simulations previously computed. The tracking information can also be used for enhanced visualization as shown in Figure 1 which depicts the trajectory of a single feature with history encoded as opacity.

With the increase in computational power available, simulation sizes are growing. These datasets are massive, and the standard post-processing feature extraction and tracking [21] approach is unfeasible because the datasets cannot be loaded onto a single processor. Automated tools to aid in searching the data such as feature extraction and tracking are even more crucial since the amount of data is too large to simply investigate let alone download for local visualization processing. To realize the full potential, the feature extraction and tracking processing must be implemented where the data resides and when it is being computed (in-situ). This motivates the need for a parallel/distributed feature extraction and tracking process that runs along with the simulation. Furthermore, one would like to interact with the tracking process while the simulation is ongoing, to change parameters, initialize the process, etc.

In [7], a distributed feature extraction algorithm was presented. In this paper, we extend the algorithm to include distributed feature tracking and integrate it within a computational steering system to support interaction while the simulation and visualization is in progress. The implementation uses GrACE (Grid Adaptive Computational Engine) [17], an infrastructure which supports distributed adaptive mesh-refinement computations on structured grids. GrACE provides multifaceted objects specialized to distributed adaptive grids and grid functions, and has been deployed to support applications in many different applications. This enables the visualization processes to run in-situ without the added overhead for data transfers. To access the ongoing simulation, the DISCOVER (Distributed Interactive Steering and COllaborative Visualization EnviRonment) portal [16] is used which provides a control network to query the analysis, interaction and steering interfaces. DISCOVER is supported by a suite of detachable interfaces and analysis modules and allows users to interact with, interrogate, control and steer GrACE-based applications. In the next section, the algorithms for both distributed feature extraction and tracking are detailed followed by the implementations. Examples are presented in Section 4.

2. DISTRIBUTED FEATURE EXTRACTION AND TRACKING

There is lot of work done in building distributed visualization systems, however most of these systems perform distributed visualization with isosurfaces or volume rendering (for example see [5][4][10]). Standard isosurface algorithms are inherently parallelize-able since they treat each cell in the grid independently and have no notion of connectivity. This is not the case here. Connected features provide much more information than simple isosurfaces, since one can compute attributes (quantifications) such as mass, volume, centroid and moments. These attributes require the knowledge of the entire feature or object [19]. While each processor can locally extract their own features, a feature and its evolution (future timesteps) may span multiple processors, so a coalescing procedure must be used to globally resolve feature identifications and compute feature attributes.

The features in [19] are defined as connected nodes that satisfy a certain criteria, such as a region where all of the nodes are above a particular threshold value. The bounding surface of this region is a standard isosurface, although now that all of the different regions are separate and distinct. The regions can be extracted with a flood fill type algorithm or with a stepped region-growing algorithm [21][19]. In addition to the nodes (data cells) within an object, the boundary polygons (isosurface), and a set of attributes are computed. Since the data is distributed among multiple processors, a connected feature may span several processors. Once each processor identifies its own local features, a "merge" must be performed to connect features spanning several processors. The merge algorithm operates by checking the boundaries between processors to see whether a particular feature can potentially overlap (i.e., hits the boundary). In [7], two different boundary merge strategies are described, a "*complete-merge"* strategy, which utilizes a binary swap algorithm [13], and a "*partial-merge"* strategy, which utilizes a *visualization accumulator*. The *complete-merge* strategy is related to the 2D parallel implementation of the component labeling (image segmentation) and watershed transformation algorithm given in [15][2]. This algorithm requires an O(log(n)) communication overhead (where n is the number of processors) to coalesce all of the features. The "*partial-merge*" is a more efficient algorithm that does not depend on the number of processors. After each processor does its own feature extraction, processors communicate with their immediate boundary neighbors to determine the local connectivity. This partial-merge data (given as a set of tables) is enough to reconstruct the full connectivity, which can be done by a *visualization accumulator* as a preprocess step to visualization. This step involves simple bookkeeping to determine that if O^1_{p1} (feature labeled 1 in processor 1) is connected to O^2_{p4} , and O^2_{p4} is connected to O^9_{p5} , then all of these pieces are part of a larger feature. A picture describing the local merge strategy is shown in Figure 2.

In addition to the "partial-merge" tables, the bounding polygons (isosurface) and other quantification measurements (volume, etc) are sent to the visualization accumulator. After the objects are merged, all of the bounding polygons pertaining to a feature are given the same color, resulting in a coherent view of the features as shown in Figure 2. Note that at the end of the processing there are 27 distinct features found. (*This data is one timestep from a Pseudo-spectral simulation of coherent*

turbulent vortex structures by Dr. N. Zabusky and Dr. V. Fernandez of the Department of Mechanical and Aerospace Engineering, Rutgers University. The dataset is 256^3 (upsampled from 128^3) with 100 timesteps.)

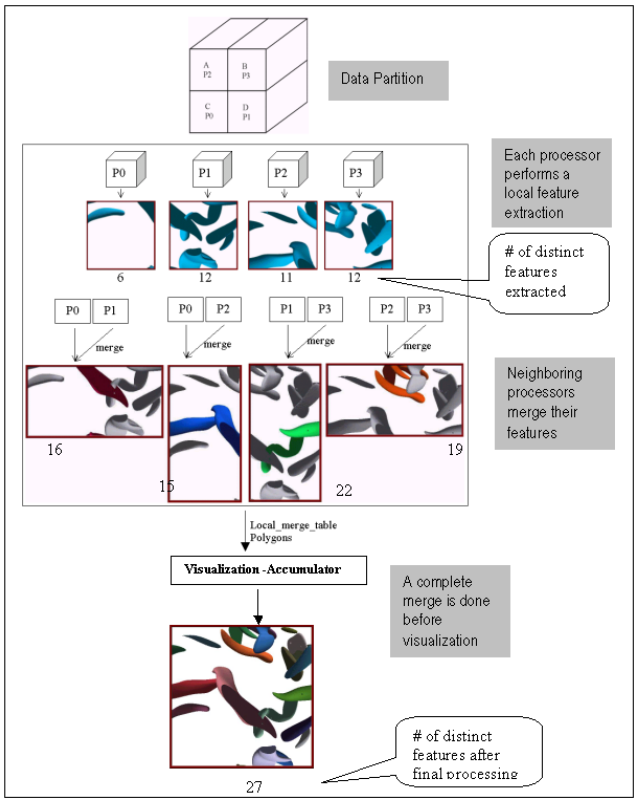

Figure 2. The "partial-merge" strategy for feature extraction.

2.1. Distributed Feature Tracking

Once features are defined, their evolution can be characterized by performing feature tracking. Feature tracking involves matching a feature in one timestep to the next timestep. Events can be categorized into one of five different classifications, i.e. *continuation, splitting, merging, creation and dissipation*. As is described in [21] the most salient characteristic of a continuing feature(s) is location. The original feature tracking algorithm worked in two phases. In the first phase, feature overlaps are determined to find features from one timestep which overlap with features from the next. This generates a set of candidates with which to perform further testing. The next step performs additional tests on the overlapping candidates to determine a "best-match". For example, if a feature in t_{i+1} overlaps with a feature in t_i by 1 voxel it is probably not the "best-match". Therefore, after overlap detection a second phase is performed which implements the best matching test to find the best correlations between features [21]. To efficiently compute overlap, the objects nodes from each timestep are sorted and the two sorted lists are "merged". As they are merged, the overlapping objects can be determined as well as the amount of overlap. The overlap is stored in a table of size *n x m*, where *m* and *n* are the number of objects in t_i and t_{i+1} respectively. Using the overlap-table, all combinations between overlapping features are computed to maximize a normalized correspondence metric [21][22].

The first phase of the feature tracking algorithm can be easily incorporated into a distributed algorithm. Assuming the each timestep has the same partition as the previous timestep, each processor can compute the overlap from one timestep to the next and determine its own overlap-table. However, the second phase of feature tracking is not as straight forward. If each processor independently computed the best match result and then merged across neighboring boundaries (as is performed in the feature extraction phase) this may not yield the correct matching results. An example is presented in Figure 3. If each processor determined its own best match, the matched result could yield that Object A and B (time t_0) combine to form Object 2 (in t_1), Object C matches to Object 3, and Object 1 is born in t_1. Simply merging the boundaries will then yield that Objects A, B and C correspond with Objects 2 and 3. When looking at all the possibilities (as is done in the sequential best match) the correct match would yield that Object A goes to Object 1 and B and C go to 2 and 3.

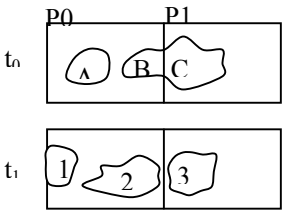

Figure 3. Determining the best match

To correct this result, a scheme similar to the local merge strategy in the feature extraction phase is utilized. Each processor computes its own overlap result and sends the table to the visualization accumulator. The viz-accumulator (in addition to computing the feature merge) can then compute the best match results. The algorithm can be summarized as shown in Figure 4. In the first stage, the local overlap detection algorithm determines the feature boundaries and interior nodes and computes the overlap from two different timesteps (t_i and t_{i+1}). The local information is sent to the viz-accumulator which uses that to compute the best match. In the second stage, viz-accumulator computes the global overlap by comparing the local overlap tables and the local merging tables (determined by feature extraction). After that, the best match algorithm is performed on the viz-accumulator.

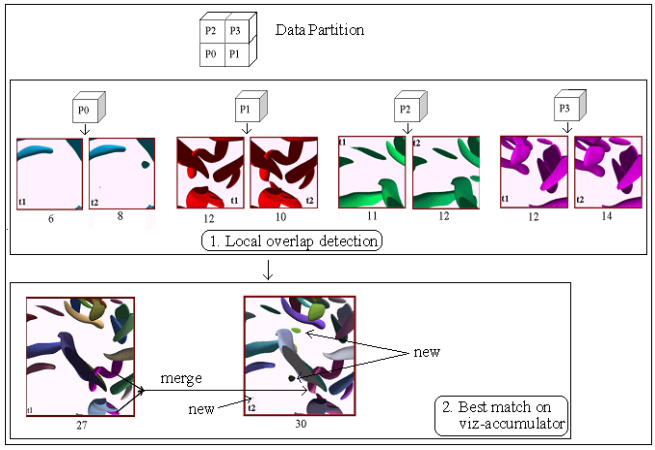

Figure 4. Parallel feature extraction and tracking example. This figure uses the same dataset as Figure 2. The number below each image is the feature count of that block data. In the final image of timestep 2, each object gets the same color as its matched feature of timestep 1.

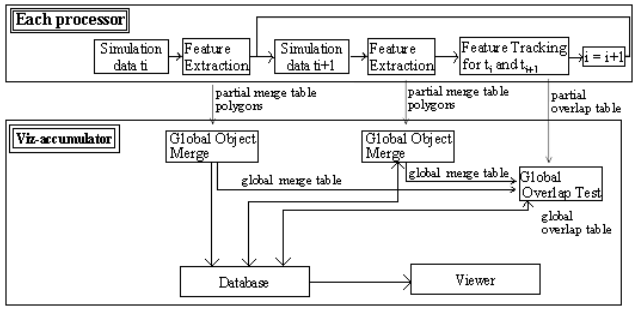

Figure 5. Overview for the distributed feature extraction and tracking system.

A full overview of both the feature extraction and feature tracking distributed processes are shown in Figure 5. Each processor determines the features and the tracking information as the data becomes available. The data is either being computed by the simulation or it is being read in by the visualization program. This is explained in more detail in the next section.

3. GRACE IMPLEMENTATION

In order for the feature extraction and tracking routines to run in-situ, the implementation must utilize the same data structure as the ongoing simulation so as to not incur any data copying overhead. For this implementation, we used the GrACE infrastructure[17]. GrACE supports distributed adaptive mesh refinement on structured grids. It contains a unified data-management substrate which transparently spans distributed processor memories as well as disks and persistent storage systems. The key feature of the data-management scheme is a natural mapping from a multidimensional, multiscale application domain to 1-dimensional storage that is directly derived from the application domain coordinate system. This mapping enables implementation of a semantically specialized virtual shared memory where the memory address space is derived from application domain coordinates. As a result heterogeneous computational and data objects can be directly accessed from virtual memory based on their location in the computational domain rather than using conventional memory addresses and pointers. The resulting data-structure implements a virtual, Hierarchical, Distributed and Dynamic Array (HDDA) spanning multiple levels of the distributed memory hierarchy.

The implementation contains three parts: (1) an initialization phase, which includes the DISCOVER communication protocols (see next section), (2) a visualization computation phase, which determines the feature extraction and tracking, and (3) a write out phase, which sends the data to the viz-accumulator (or through the DISCOVER portal). Because the HDDA grid abstraction has been optimized for ghost communications, the neighbor processor communication can be done using ghost regions. After feature extraction, on each processor local object information (such as local feature volume/mass, feature id etc.) of those features which contain boundary cells will be written into a Grid Function. A ghost region synchronization of that grid function will be performed to pass the local information to direct neighbors. Each processor can then determine the local feature merge table by comparing the neighboring feature information in the ghost region with its local features to see if any of them are potentially connected. This process is similar with the partial merge strategy in the distributed feature extraction section. The local feature merge information will be passed to the viz-accumulator while simulation finishes to completion.

The GrACE implementation works with both in-situ data or data that is currently being computed (during the simulation) or as a post-processing step for just visualization where the data has already been computed and is stored on disk. Both types of examples are shown in Section 4.

3.1. Discover Portal and Visualization

DISCOVER is a generic framework that enables interactive steering of scientific applications and also allows for collaborative visualization of data sets generated by such simulations [1]. DISCOVER is supported by a suite of detachable interfaces and analysis modules and allows users to interact with, interrogate, control and steer GrACE-based applications through a web based portal. A snapshot of the GrACE interface portal is shown in Figure 6. In this work, we have included some DISCOVER controls within the feature extraction and tracking process which enables a user to start or stop the feature tracking and to change the

threshold setting. This is especially useful for long simulations where interesting events may not take place until much later in the simulation.

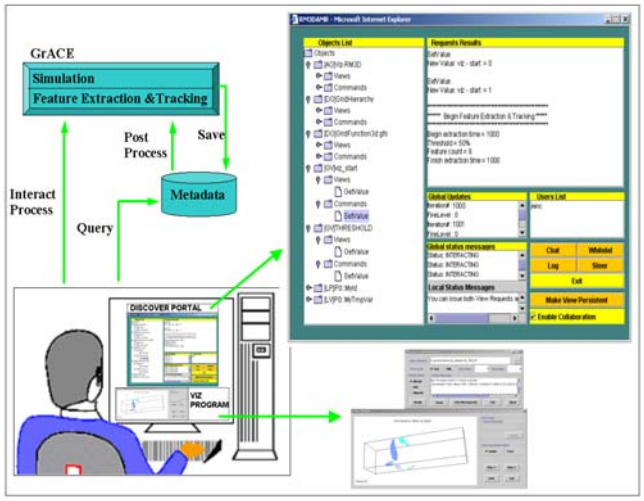

Figure 6. In-situ feature extraction and tracking with DISCOVER

The feature extraction and tracking routines produce files which can then be loaded for visualization. The files contain the tracked results (as a set of graphs) and associated polygonal data containing isosurfaces. This information is read into the viz accumulator which is a bookkeeping program designed to correlate the various files and perform best matching. The final result can then be visualized using AVS express or VTK modules written for this type of data [2]. (These codes are available on our website[2]).

4. RESULTS

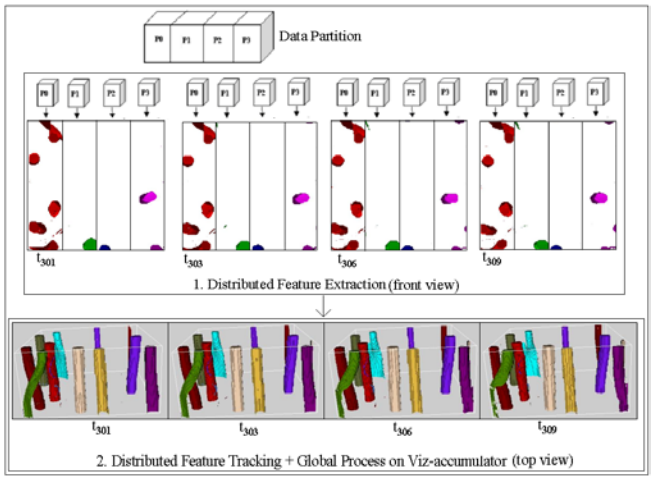

Figure 7. Distributed feature extraction and tracking example

We applied the distributed feature extraction and tracking algorithm to three different simulations. The first two utilized feature extraction and tracking as a post processing step, i.e., to process the visualization. The first simulation was a pseudo-spectral simulation of coherent vortex structures. The data was 256^3 with 100 timesteps. Figures 2 and 4 show results from this dataset. Figure 7 contains four timesteps from a simulation of rotating, stratified turbulence using the quasi-geostrophic (QG) equations [8][9]. The dataset is 960x960x480 with 1000 timesteps. In this figure, the features in each processor as well as the merged result are shown. Note how the isosurfaces from each distinct timestep are colored appropriately. (Both of these datasets were upsampled from lower resolutions to demonstrate different partition schemes.)

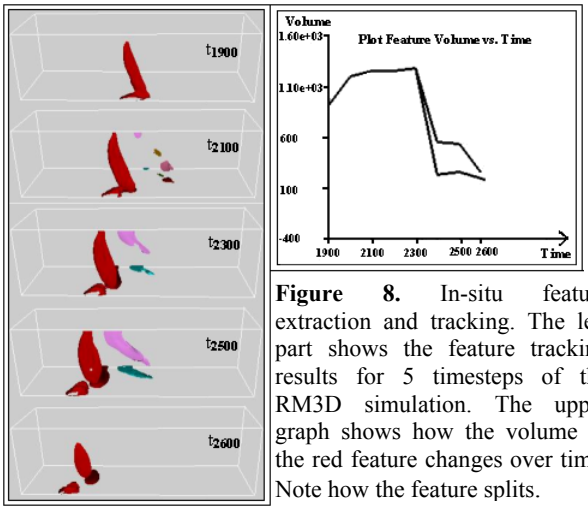

Figure 8. In-situ feature extraction and tracking. The left part shows the feature tracking results for 5 timesteps of the RM3D simulation. The upper graph shows how the volume of the red feature changes over time. Note how the feature splits.

The third example is using *in-situ* feature extraction and tracking. The simulation, called RM3D, uses a compressible Euler equation for shock accelerated inhomogeneous flows (Richtmyer-Meshkov) [20]. In this environment, baroclinic vorticity is playing the major role of hydrodynamic instability and late time turbulent mixing. 8000 timesteps were run with a resolution of 256x64x64, with a threshold of 50%. Figure 8 displays 5 timesteps from between timesteps t=1900 and t=2600. In this simulation, the large red feature splits into two separate features (the children are given the same color as the parent, however, other coloring options are also available [21]). The graph on the right displays the computed quantification, i.e. the volume of the red object over these timesteps. The break in the graph depicts where the feature split. In this example, the DISCOVER portal was used to control the simulation and start the feature tracking at timestep 1000.

5. DISCUSSION

The implementation presented here can be used with any application implemented with GrACE. The methodology can also be incorporated into any other distributed infrastructure used for compute intensive simulations (such

as CHOMBO [3]). For these examples, no load balancing was performed. For the in-situ mode, load balancing is done by the simulation and should be equivalent for the visualization processing, because the "features" are most those locations that need to be load balanced for the simulation. We are currently investigating the load balancing work for the post-processing mode.

Another area of investigation is AMR. For adaptive meshes, individual portions of the data may contain higher resolution data at different levels. Since GrACE contains support for adaptive mesh refinement, the distributed feature extraction and tracking algorithm presented here can easily incorporate the various levels by creating extra bookkeeping files which can then be coalesced by the viz-accumulator. The visualization viewers will also have to be accommodated to support the new queries which will now be possible (i.e. showing connected regions through levels and time).

Another area of investigation is to incorporate different types of "feature", and to extend the feature tracking to include a "contour tree" like representation [6].

6. CONCLUSIONS

For massive time-varying simulation, feature analysis and visualization tools are crucial to help interpret all of the resulting datasets. In this paper, we presented an algorithm and implementation of a distributed system for in-situ feature extraction and tracking of massive datasets. The implementation could be used as a distributed visualization algorithm or as a system that runs along with a computing simulation. We have also demonstrated the ability to interact with the ongoing simulation.

This work was done at the Vizlab, CAIP Center, Rutgers University. We gratefully acknowledge the support of the National Science Foundation (ITR 0082634). We would also like to thank Shuang Zhang for his help with the RM3D and RM2D simulations.

REFERENCES

[1] GrACE and DISCOVER homepage: http://www.caip.rutgers.edu/TASSL/

[2] Vizlab homepage: http://www.caip.Rutgers.edu/vizlab.html.

[3] CHOMBO: http://seesar.lbl.gov/ANAG/chombo.index

[4] ChomboVis: http://seesar.lbl.gov/ANAG/chombovis.html

[5] *Proceedings of the IEEE Symposium on Parallel and Large Data Visualization (PVG)*, October 2001 (also 1999,1997). ACM Press

[6] C. Bajaj, A. Shamir and B.-S. Sohn, Progressive Tracking of Isosurfaces in Time-Varying Scalar Fields, *CS & TICAM Technical Report, University of Texas at Austin,* 2002.

[7] J. Chen, Y. Kusurkar, and D. Silver, Distributed Feature Extraction. *Proc. SPIE Vol. 4665*, p.189-195, Visualization and Data Analysis 2002, 3/2002.

[8] D. Dritschel and M. Ambaum, A Countour-Advective Semi-lagrangian Numerical Algorithm for Semulating Fine-Scale Conservative Dynamical Fields. *QJRMS*, April 1997.

[9] D. Dritschel and M. Torre Juarez, The Instability and Breakdown of Tall Columnar Vortices in a Quasi-Geostrophic Fluid. *Journal of Fluid Mechanics*, Aug 1996.

[10] L. Durbeck, N.Macias, D.Weinstein, C.Johnson and J.Hollerbach, SCIRun/Haptic Display for Flow Fields. *Third PHANToM User's Group Workshop*, Oct 1998.

[11] B. Guo. Interval Set: A Volume Rendering Technique Generalizing Isosurface Extraction. *In Proceedings IEEE Visualization '95*, pages 3-10, Atlanta, Georgia, October 29-November 3 1996.

[12] W. Koegler, Case Study: Applications of Feature Tracking to Analysis of Autoignition Simulation Data. *IEEE Visualization (Vis 2001)*, San Diego, CA, Oct 2001.

[13] Y. Kusurkar, Distributed Feature Extraction, MS Thesis, Rutgers, The State University of New Jersey, October 2000.

[14] Moga, and M. Gabbouj, Parallel Image Component Labeling with Watershed Transformation. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, May 1997, Vol 19, No 5.

[15] Moga, Parallel Multiresolution Image Segmentation with Watershed Transformation. *ACPC'99, LNCS 1557*, P. Zinterhof, M. Vajtersic, and A. Uhl Editors, Springer Verlag 1999, pp. 226-235.

[16] R. Muralidhar, M. Parshar, An Interactive Object Infrastructure for Computational Steering of Distributed Simulations. *Proceedings of the Ninth IEEE International Symposium on High-Performance Distributed Computing (HPDC 2000)*, IEEE Computer Science Society Press, pp. 304-305, Aug 2000.

[17] M. Parashar and J. Brown, Distributed Dynamic Data-structures for Parallel Adaptive Mesh-Refinement. *HiP*C, December 1995.

[18] S. Park, C. Bajaj and I. Ihm, Effective Visualization of Very Large Oceanography Time-Varying Volume Dataset. *CS & TICAM Technical Report*, University of Texas at Austin, 2001.

[19] F. Post and T. van Walsum and F. H. Post and D. Silver. Iconic Techniques for Feature Visualization. *In Proceedings of IEEE Visualization* '95, pages 288--295,Atlanta, Georgia, October 1995.

[20] R. Samtaney and D. I. Pullin, On initial--value and self--similar solutions of the compressible Euler equations. *Phys. Fluids*, Vol 8(10), pp:2650-2655, 1996.

[21] D. Silver and X. Wang. Tracking and Visualizing Turbulent 3D Features. *IEEE Transactions on Visualization and Computer Graphics*, 3(2), June 1997.

[22] D. Silver and X. Wang, Tracking Features in Unstructured Datasets. *Proceedings of IEEE Visualization '98 Conference*, October 1998, Research Triangle Park, NC.

[23] D. Silver. Object-Oriented Visualization. *IEEE Computer Graphics and Applications*, 15(3), May 1995.

[24] T.v. Walsum. Selective Visualization on Curvilinear Grids. *Ph.D thesis, Delft University of Technology*, Delft, The Netherlands, 1995.