Scalable Parallel Tracking of Time-Varying 3D Flow Features

Authors Name/s per 1st Affiliation (Author)
line 1 (of Affiliation): dept. name of organization
line 2: name of organization, acronyms acceptable
line 3: City, Country
line 4: Email: name@xyz.com

Authors Name/s per 2nd Affiliation (Author)
line 1 (of Affiliation): dept. name of organization
line 2: name of organization, acronyms acceptable
line 3: City, Country
line 4: Email: name@xyz.com

***Abstract*—Large-scale time-varying volumetric data set can take terabytes to petabytes of storage space to store and process. One promising approach is to process the data in parallel in situ or batch, extract features of interest and analyze only those features, to reduce required memory space by several orders of magnitude for the following visualization tasks. However, extracting volumetric features in parallel is a non-trivial task as features might span over multiple processors, and local partial features can be only visible within their own processor. In this paper, we discuss how to generate and maintain connectivity information of features across different processors. Based on this connectivity information, partial features can be integrated, which makes possible for the large-scale feature extraction and tracking in parallel. We demonstrate the effectiveness and scalability of different approaches on two data sets, and discuss the pros and cons of each approach.**

***Keywords*-feature tracking; parallel graph;**

I. Introduction

The accessibility to supercomputers with increasing computing power has enabled scientists to simulate physical phenomena of unprecedented complexity and resolution. However, these large-scale time-varying data sets can take giga- or even tera-bytes of space to preserve. One promising solution to the problem is to introduce feature extraction and tracking techniques. Instead of duplicating raw data along the exploring process, dealing with features of interest typically requires memory space that is several orders of magnitude smaller than the raw data. However, large vortical data sets are typically presented and processed in a distributed fashion simply because of the shear size. Extracting and tracking features embedded in a distributed vortical data set is a non-trivial task. Existing researches on feature-based data visualization have been done mostly focusing on decomposing features using quantitative measures such as size, location, shape or topology information, and so on. These measures cannot be applied to distributed volume data directly since vortical features consist of certain amount of voxels and are very likely to span over distributed data blocks as they evolve over time. Therefore existing quantitative measures of partial data scattered across different data blocks cannot be used to describe an integrated feature, unless the distribution of partial features can be obtained beforehand.

To obtain the distribution information of features, a connectivity map of each feature should be generated and maintained. In this paper, we propose an approach to creating and maintaining feature residual information as well as connectivity information using parallel graphs. Comparing to the existing approaches, which generate and maintain the global feature information in a single host node, our approach can be done locally that only involves residual data blocks of target features. This requires least communication overhead and avoids the potential link contention. We demonstrate the effectiveness of this method with three vortical flow data sets and the scalability of our system in a distributed environment.

II. Related Work

Extraction and tracking are two closely related problems in feature-based visualization. Although many feature tracking algorithms have been introduced, most of them extract features from individual time steps and then try to associate them between consecutive time steps. Silver and Wang [26] defined threshold connected components as their features, and tracked overlapped features between successive time steps by calculating their differences. Octree was employed in their method to speed up the performance and the criteria they used were domain dependent. Reinders et al. [24] introduced a prediction verification tracking technique that calculates a prediction by linear extrapolation based on the previous feature path, and a candidate will be added to the path if it corresponds to that prediction. Ji and Shen [17] introduced a method to track local features from time-varying data by using higher-dimensional iso-surfacing. They also used a global optimization correspondence algorithm to improve the robustness of feature tracking [16]. Caban et al. [3] estimated a tracking window and then tried to find the best match between that window and different sub-volumes of subsequence frames by comparing their distance of textural properties. Also, Bremer et al. [1] described two topological feature tracking methods, one employs Jacobi sets to track critical points while another uses distance measures on graphs to track channel structures.

Most of the aforementioned methods extract features from each time step independently and then apply the correspondence calculation. This could become very slow when the size of data becomes large. Muelder and Ma [19] introduced a prediction-correction approach that first predicts feature region based on the centroid location of previous time steps, and then corrects the predicted region by adjusting the surface boundaries via region growing and shrinking. This approach is appealing because of its computing efficiency and the reliability in an interactive system.

From the perspective of parallel processing, most of the existing feature extraction and tracking approaches rely on certain forms of global feature information which, however, must be obtained via communication in a distributed environment. Chen and Silver [] introduced a two stage partial-merge strategy using the master-slave paradigm. The slaves first exchange local connectivity information using Binary-tree merge, and then a visualization host collects and correlates the local information to generate the global connectivity. This approach is not scalable since half of the processors will become idle after each merge. It is also unclear how the visualization host can efficiently collect local connectivity information from non-server processors, since gathering operation is typically very expensive given a large number of processors.

The approach proposed in this paper follows a different strategy. Instead of sending local connective information back to the host, the local connectivity information is computed and preserved only in the nodes where correspondent features reside. There is no global connectivity information preserved in the host, and the host only serves as an interface to broadcast the criterion of feature of interest to the other nodes. In this way, the computation of merging local connectivity information is distributed to the slaves and hence effectively reduces the potential communication bottleneck on the host. Moreover, our approach does not need a global synchronization for gathering all local connectivity information at the host, and thus avoid potentially long computation time caused by features spanning over a large number of nodes. This makes it ideal for an interactive system, where users can select the features of interest and instantly receive the visual feedback as the features evolves.

[TODO] Add related work for parallel graph algorithms and data structures. (half page)

III. Algorithm Design

Though features can be extracted within individual processors using the conventional methods, they might also span over multiple data blocks, which is unavoidable as the number of processors increases. To extract and trace a feature over a distributed volume data set, we need to build and maintain the connectivity information of the feature across multiple nodes. As feature descriptors, such as size, curvature, and velocity, are distributed among processors, connectivity information can facilitate us to obtain the description of a feature from neighboring processors, and enable more advanced operations such as similarity evaluation. However, it typically requires data exchanges among processors to build and maintain the connectivity information, and thus incurs extra communication cost. To design a proper communication scheme for better performance and scalability, we carefully consider the following three factors in our design:

- How many communications are required to complete the connectivity graph;
- How many processors will be involved in one communication;
- How much data will be sent and received in one communication.

In the following sections, we will give a detailed description on how to build and maintain such connectivity information using an undirected unweighted graph to minimize the cost over the above three factors.

A. *Feature Extraction*

Feature extraction is a process that first detects a feature, and then calculates quantitative attributes characterizing the feature. In general, a feature can be any interesting pattern, structure, or object that is considered relevant for investigation. In our application, a feature is defined as the collection of voxices encompassed by a certain iso-surface. Such a volumetric feature could be extracted by conventional techniques such as region growing, geometry or topology based clustering, or other domain specific algorithms as described in [] [] [] [].

In our work, we use a standard region-growing algorithm [14] to partition the original volume data into an initial set of features. This can be done by first spreading a set of seeding points inside the volume, and then clustering voxices into separate regions. Each region is regarded as a single feature. Usually, scientists prefer to focus on features that are meaningful as dictated by their underlying research studies. Potential features detected by the region-growing algorithm, however, are often cluttered and contain noise, which might be far from expected. Therefore, an iterative refinement process is conventionally applied to improve the results after the potential feature set being generated. On the other hand, we note that the refinement process can typically reduce the number of features, which also can lower computational cost of tracking.

Tangelder and Veltkamp [27] presented the applicability and performance of context-based 3D shape descriptors and the corresponding retrieval methods. Their work suggested that when designing an interactive refinement process, we need to carefully select a set of feature descriptors, which should be deformation robust, such as size, location, or skeletal curvature. In addition, we also need to consider computational efficient in our design. After specific features have been identified from a single time step, we can track their evolution over time using a prediction-correction approach []. The feature locations in a future time step can be predicted from the location of features in the previous time steps. Once prediction is made, the actual region can be obtained by adjusting the surface of the predicted region: first shrink the edge surface points to obtain the mutual region between consecutive time steps, and then use of a region-growing method to generate the refined region.

This prediction-correction approach was proved effective and efficient for tracking feature on a single processor [32]. However, when the size of the volume data becomes too large to be fit into one processor, a cluster of multiple processors are often needed to process the data in parallel. One challenge to parallelize the tracking approach lies in that it can be difficult to obtain the global feature descriptions unless they can be shared and merged in an efficient way. This is because volumetric features may span over multiple processors and partial features on different nodes are operated independently.

An intuitive way of exchanging such feature information is first to find how features span over multiple processors, and then to merge feature descriptions according to the connectivity information.

B. *Creating Partial Connectivity Graph*

If we partition a volumetric data set into the blocks across a regular processor grid, it is very likely that some features can cross multiple processors and blocks. We note that the cross-section of such a feature on the boundary of two adjacent blocks should match. Leveraging this property, we could connect separate parts of a feature on the adjacent data blocks by comparing their sections on the correspondent boundary surface.

Since data is distributed, each processor is not aware of the partial features identified on the other processors. Exchanging the voxels on the sectional area between two adjacent processors would be sufficient for finding the possible matches of partial features between these two processors. However, we choose to exchange more abstract data, that is the minimal and maximal (min-max) coordinate values and the geometric centroid of the cross-sectional area, to reduce the amount of data exchanged over network. The min-max coordinate values are not optional because they ensure correct connectivity for some special cases where one cross-sectional area is surrounded by another concave or hollow area whose centroid points happen to be the same, as depicted in Figure 1.

A voxel-width "ghost surface" that stores boundary surface belonging to neighboring blocks might help to achieve voxel-wise matching for partial features. However, maintaining such ghost surfaces requires frequent inter-process communication and is considerably expensive for data generated

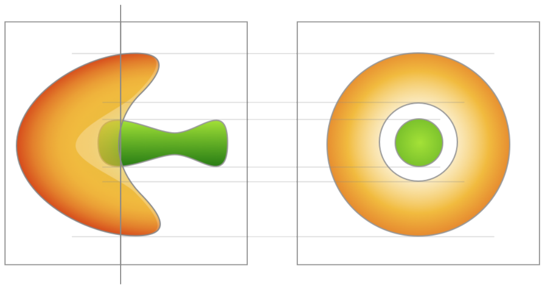

Figure 1. A special case where two features share a same centroid on the section.

in real-time. Instead, we use a simplified approach which requires much less communication cost. That is, if the spatial distance between two centroid points of two neighboring partial features is less than two voxels, the two features are considered a match and belong to the same feature.

The creation of partial connectivity graph will not introduce extra computation cost as it can be done along with the direction of region growing. A new edge is appended to the existing graph when a local feature touches the block boundary. In the graph, the new edge connects one existing block with the new detected block of a feature. The ending vertices of an edge is denoted using the block indices. Thus the ID of the new detected block is assigned as one ending vertex of the new edge. In addition, the global index of the centroid point is assigned as the edge value.

Algorithm 1 shows the detailed algorithm of creating local partial connectivity graph.

// TODO (put to result, performance analyze) Memory cost, each feature on boundary will use two INTs, 1 as global centroids coordinate, the other as target node number. even if there's 1000 features on the boundary, it would cost more than 1000 * 5 * 4 = 20mb to store this graph, neglectable compare to the volume data itself. (But kind of large if used for communication)

C. *Creating Global Connectivity Graph*

1) The naive solution: Local connectivity graphs need to be merged to construct the global description of a partitioned feature. A naive approach to gathering local connectivity graphs is to let each processor exchanges the shared edges with its neighbors. After exchanges, two edges are merged at a processor if they satisfy the following three conditions:

- The starting and ending vertices are reversely matched;
- The min-max boundary coordinate do match;
- Edge centroid is located within direct neighbors.

Recall that the starting vertex of an edge represents the current processor ID, and the ending vertex represents the ID of a neighboring processor whose partial feature is adjacent to the local one. The edge value represents the global coordinate of the geometric centroid and the min-max boundary on the shared boundary surface. If two edges

Algorithm 1 Creating Partial Connectivity Graph

```
1:  $edgeList \leftarrow newList()$
2:  $featureList \leftarrow newList()$
3:  if Current time step $t = t0$ then
4:      // Initialize random seeding points
5:      $seedList \leftarrow randomVortices()$
6:      for each $seed$ in $seedList$ do
7:          $feture \leftarrow expendRegion()$
8:          append $feture$ to $featureList$
9:      end for
10: else
11:     for each $feture$ in $featureList$ do
12:         $feture \leftarrow predictRegion()$
13:         $feture \leftarrow adjustRegion()$
14:         $start \leftarrow$ current processor ID
15:         $end \leftarrow$ target neighboring processor ID
16:         $min, max \leftarrow$ min-max boundary coordinate
17:         $index \leftarrow$ global voxel index of centroid
18:         Edge $e = Edge(start, end, min, max, index)$
19:         append $e$ to $edgeList$
20:     end for
21: end if

1:  $adjustRegion()$
2:  if Voxel $v$ on boundary surface then
3:      $updateMixMaxBoundary()$
4:      $updateBoundaryCentroid()$
5:  end if
```

match to each other, the two partial features must share a same boundary section with the same centroid and section region. In other word, these two partial features are resulted by partitioning a original feature, and should be considered the same feature sharing a same feature ID.

Algorithm 2 shows the detailed process to merge matched edges (henceforth referred to as *REDUCE*).

Algorithm 2 REDUCE: Merging Matched Edges

```
Require: $localEdges$, $recievedEdges$
1:  for each $ei$ in $localEdges$ do
2:      for each $ej$ in $recievedEdges$ do
3:          if $ei.start = ej.end$ and $ej.start = ei.end$ and
            $ei.min = ej.min$ and $ei.max = ej.max$ and
            $ei.centroid \approx ej.centroid$ then
4:              if $ei.id < ej.id$ then
5:                  $ej.id \leftarrow ei.id$
6:              else
7:                  $ei.id \leftarrow ej.id$
8:              end if
9:          end if
10:     end for
11: end for
```

This naive solution may work for data sets with small features. However, if there are long curly features partitioned evenly over the process grid, each processor needs to consecutively communicate with its neighbors to inclemently gather and merge a global feature. This requires O(N_p) communications to connect a single feature, where N_p is the number of processors in the grid. Consequently, the total communication cost is O($N_f \times N_p$) times communication, where N_f is the number of features within one time step. In addition, since one processor cannot predict how many edges it will receive from its adjacent processors, it is hard to schedule the communication process.

2) The centralized approach: A possible solution is to use a master-slave hierarchy to reduce the number of communication required for merging all edges. The master-slave hierarchy can be constructed using a separate host processor. When the feature extraction process is done, the edges of a local graph represents the number of features across the block boundary, and also encodes the location information of each local feature. All local graphs are then gathered to the host processor. After this *Gather* operation is done, that is the host processor has collected all local graphs, the REDUCE operation starts to merge the edges from each partial graph to construct a single global connectivity graph.

The merit of this centralized approach lies in that it requires inter-processor communication only once. Moreover, the global graph of feature information can be preserved in the host, and the host can response feature queries directly without pulling information the slaves again. However, this approach has an obvious drawback. Since all partial graphs are sent to the host, there exists potential bottlenecks, both in communication and computation, on the host.

3) The decentralized approach: A better solution is to decentralize from REDUCE on a single host processor to all processors that are available. After the feature extraction process is done and so does the creation of local connectivity graphs, an *Allgather* process starts to exchange all local connectivity graphs within each processor to all the others. Each processor will first collect a full copy of all local connectivity graphs followed by the same REDUCE process to merge the edges into a single concise connectivity graph.

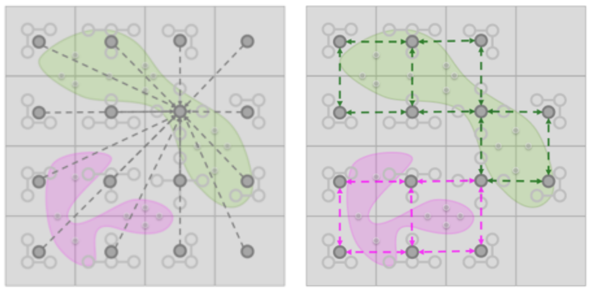

Figure 2. Left: each processor gather partial graphs from all the other processors; Right: a special case where two features share a same centroid.

where is the figure 2 referred?

Though the “redundant” host processor is no more required when applying for computation resources, this approach does not actually solve the bottleneck problem since now every processor is now acting like a host since they still need to gather all partial graphs and try merging them together. For real world data set, however, it is rarely the case that a single feature will span over every processor. In other word, it is unnecessary for a processor to gather edges of features that are not local. To reduce the redundant communications with every other processor in the grid, the decentralized approach could be further improved such that we only consider those processors that are directly adjacent to the current one. That is, each processor only talks to its direct neighbors and shares information with them regardless what is happening outside.

For a regularly partitioned volumetric data set, there are at most six direct neighbors for each processor. Instead of gathering connectivity information from all other processors in the grid, each processor only gathers local graphs of neighboring processors. This could be considered as a higher level of region growing, which starts from one seeding processor and then grows to its adjacent processors by exchanging and merging connectivity information in a bread-first fashion until all cross-boundary features are connected.

The detailed scheduling algorithm is depicted in Algorithm 3.

what is this?

For the worst case that a feature spans over all of the processors, it takes O(C(—N—)) time to finish the search for connected processors, much faster than the previous O(N) time for all gathering among all processors.

4) The hybrid approach: We can still take a step further to optimize the aforementioned decentralized approach. As volume data evolves over time, the internal features may vary but should not change drastically in size and shape nor location if the time interval for sampling is sufficiently small. Thus we can apply the prediction-correction approach to further reduce the number of times required to complete the connectivity graph.

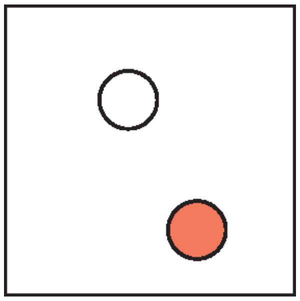
Figure 3.

For every time step, t_i, when the global connectivity graph is completed, new local communicators will be updated for the next time step, t_{i+1}, with the union of processors that share a same edge with the current processor, as shown in Figure 3. However,the edges from these processors are required to complete the global connectivity graph, no matter which approach is used. Hence, for these must-involve processors, we apply the Decentralize-I approach, allowing the minimum one-time synchronization to finish gathering all edges that are necessary for updating the connectivity graph based on the graph created at the previous time step, t. Then, the processor-level region growing explained in 3.3.3 is applied to extend the “boundary” of processes, obtaining newly connected processors caused by the evolution of the original volume. In addition, only those edges that are changed and not synchronized will be sent. This ensure that we minimize the amount of data being sent over network. The detail algorithm of the hybrid approach is given in Algorithm 4.

Algorithm 3 Processor Level Region Growing

```
Require: $adjacentProcessors, localEdges$
 1: $toSend, toRecv \leftarrow true$ // init scheduling flags
 2: $\delta \leftarrow localEdges$ // init data to be sent
 3: while $toSend$ = $true$ or $toRecv$ = $true$ do
 4:   $target \leftarrow toRecv$ = $true$ ? $myRank$ : null
 5:   $procsToSync \leftarrow Allgather(target)$
 6:   for each $proc$ in $procsToSync$ do
 7:     if $toSend$ = $true$ then
 8:       send $\delta$ to $proc$
 9:     end if
10:     if $toRecv$ = $true$ then
11:       receive $\delta\prime$ from $proc$
12:     end if
13:   end for
14:   $toSend \leftarrow procsToSync$ is empty ? $false$ : $true$
15:   $toRecv \leftarrow false$
16:   $localEdges \leftarrow Reduce(localEdges, \delta\prime)$
17: end while

 1: $Reduce(localEdges, \delta\prime)$
 2: for each $edge$ in $\delta\prime$ do
 3:   if $edge$ is new then
 4:     add $edge$ to $\delta$
 5:     $toRecv \leftarrow true$
 6:   end if
 7: end for
```

Algorithm 4 TBA

```
 1:
```

IV. Application

A. Feature Selection and Refinement

To allow a user to select or highlight certain features is commonly required in various applications. Since the feature extraction is performed independently within each PE, we need to design a method to let each processor know if the features on its adjacent processor are selected or not. Intuitively, we can implement this by sending a message to the adjacent PE whenever a feature was detected to touch the surface boundary. But if the target feature spans over multiple PEs, this sending/receiving procedure would take several rounds to end. This is potentially a big problem when the PE/Volume ratio is relatively high such that each feature spans over a lot of PEs. Another problem is that, if a PE has two selected features whose connectivity information arrive in different rounds, it requests to compute twice, which again, will become a problem when PE/Volume ratio is large.

By introducing the global connectivity graph in our approach, whenever part of the feature was selected, the unique feature id will be sent back to the host processor and then be broadcasted to all PEs containing it. Thus, the selection can be finished only in one round.

Based on the coordinates user specified or clicked on the volume rendering result, we can identify the user selected processor and feature. Then the host processor imply broadcasts the selected feature id to all those processors who has the partial feature to be highlighted.

B. Feature Tracking

Before you begin to format your paper, first write and save the content as a separate text file. Keep your text and graphic files separate until after the text has been formatted and styled. Do not use hard tabs, and limit use of hard returns to only one return at the end of a paragraph. Do not add any kind of pagination anywhere in the paper. Do not number text heads-the template will do that for you.

Finally, complete content and organizational editing before formatting. Please take note of the following items when proofreading spelling and grammar:

V. Result

A. Performance Result

B. Visualization Result

VI. Conclusion

In this paper, we proposed a decentralized approach that all feature connectivity information are created and preserved among distributed processors. Traditional approaches perform connectivity test on each processor and subsequently correspond them in a host processor after gathering all or partially merged connectivity information. Our approach does not follow this paradigm. Rather, instead being sent back to the host, the local connectivity information are computed and preserved only in the local processor. There is no copy of the global feature information preserved in the host, and the host only acts as the interface from where the criterion of feature of interest is broadcast. In this way, the computation of merging local connectivity information is distributed to the slaves, which can effectively remove the potential communication bottleneck on the host processor. Moreover, there's no need to set a barrier and wait for all connectivity information being sent back to the host, thus if one of the features spans over a large number of processors but was not selected by the user, the potentially long computation time for this feature will not be considered. This makes it ideal for an interactive system, where users can select the feature of interest and instantly receive the visual feedback as the feature evolves. unreadable it seems to repeat the previous sentences.

Acknowledgment

The authors would like to thank... more thanks here

References

[1] P.-T. Bremer, E.M. Bringa,M. A. Duchaineau, A. G. Gyulassy, D. Laney, A. Mascarenhas, and V. Pascucci. Topological feature extraction and tracking. Journal of Physics: Conference Series, 78(1):712, 2007.

[2] X. T. C. Garth. Topology- and feature-based flow visualization: Methods and applications. SIAM Conference on Geometric Design and Computing, 2005.

[3] J. Caban, A. Joshi, and P. Rheingans. Texture-based feature tracking for effective time-varying data visualization. IEEE Transactions on Visualization and Computer Graphics, 13(6):14721479, 2007.

[4] N. D. Cornea, D. Silver, and P. Min. Curve-skeleton applications. In Proceedings of the IEEE Visualization Conference, pages 1321, 2005.

[5] N. D. Cornea, D. Silver, and P. Min. Curve-skeleton properties, applications, and algorithms. IEEE Transactions on Visualization and Computer Graphics, 13(3):530548, 2007.

[6] J. Ebling and G. Scheuermann. Clifford convolution and pattern matching on vector fields. In VIS 03: Proceedings of the 14th IEEE Visualization 2003 (VIS03), page 26, 2003.

[7] J. Ebling and G. Scheuermann. Clifford fourier transform on vector fields. IEEE Transactions on Visualization and Computer Graphics, 11(4):469479, 2005.

[8] J. Ebling and G. Scheuermann. Clifford Convolution and Pattern Matching on Irregular Grids. Springer Berlin Heidelberg, 2006.

[9] H. Hauser, H. Hagen, and H. Theisel. Topology-based Methods in Visualization. Springer Publishing Company, Incorporated, 2007.

[10] H.-C. Hege, K. Polthier, and G. Scheuermann. Topology-Based Methods in Visualization II. Springer Publishing Company, Incorporated, 2008.

[11] E. Heiberg, T. Ebbers, L. Wigstr, and M. Karlsson. Three-dimensional flow characterization using vector pattern matching. IEEE Transactions on Visualization and Computer Graphics, 9:313319, 2003.

[12] J. L. Helman and L. Hesselink. Representation and display of vector field topology in fluid flow data sets. Computer, 22(8):2736, 1989.

[13] H. Helwig. Visual analysis of deferential information. In Proceedings of the International Conference of Applied Mathematics, 2006.

[14] R. Huang and K.-L. Ma. Rgvis: Region growing based techniques for volume visualization. In Proceedings of Pacific Graphics 2003 Conference, pages 355363, October 2003.

[15] J. Jeong and F. Hussain. On the identification of a vortex. Journal of Fluid Mechanics, pages 6994, June 1995.

[16] G. Ji and H.-W. Shen. Feature tracking using earth movers distance and global optimization. In Pacific Graphics, 2006.

[17] G. Ji, H.-W. Shen, and R. Wenger. Volume tracking using higher dimensional isosurfacing. In Proceedings of the IEEE Visualization Conference, pages 209216, 2003.

[18] M. Jiang, R. Machiraju, and D. Thompson. Detection and visualization of vortices. In The Visualization Handbook, pages 295309, 2005.

[19] C. Muelder and K.-L. Ma. Interactive feature extraction and tracking by utilizing region coherency. In Proceedings of IEEE Pacific Visualization 2009 Symposium, April 2009.

[20] K. Palagyi and A. Kuba. A parallel 3d 12-subiteration thinning algorithm. Graph. Models Image Process., 61(4):199221, 1999.

[21] S. W. Park, B. Budge, L. Linsen, B. Hamann, and K. I. Joy. Multidimensional transfer functions for interactive 3d flow visualization. In PG 04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference, pages 177185, 2004.

[22] A. Pobitzer, R. Peikert, R. Fuchs, B. Schindler, A. Kuhn, H. Theisel, K. Matkovic, and H. Hauser. On the way towards topology-based visualization of unsteady flow the state of the art. In accepted for Eurographics 2010 - State of the Art Reports, April 2010.

[23] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch. The state of the art in flow visualization: Feature extraction and tracking. Comput. Graph. Forum, 22(4):775792, 2003.

[24] K. F. Reinders, K. Frederik, and J. Reinders. Feature-Based Visualization of Time-Dependent Data. PhD thesis, 2001.

[25] M. Schlemmer, M. Heringer, F. Morr, I. Hotz, M. Hering-Bertram, C. Garth, W. Kollmann, B. Hamann, and H. Hagen. Moment invariants for the analysis of 2d flow fields. IEEE Transactions on Visualization and Computer Graphics, 13(6):17431750, 2007.

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

[27] J. W. H. Tangelder and R. C. Veltkamp. A survey of content based 3d shape retrieval methods. In SMI 04: Proceedings of the Shape Modeling International 2004, pages 145156, 2004.

[28] O. Amoros, S. Escalera, and A. uig. Adaboost GPU-based Classifier for Direct Volume Rendering. GRAPPSciTePress (2011), p. 215-219.

[29] The State of the Art of Flow Visualization: Feature Extraction and Tracking.