Hierarchical Streamline Bundles

Hongfeng Yu, *Member, IEEE,* Chaoli Wang, *Member, IEEE,*
Ching-Kuang Shene, *Member, IEEE Computer Society,* and Jacqueline H. Chen

Abstract—Effective three-dimensional streamline placement and visualization plays an essential role in many science and engineering disciplines. The main challenge for effective streamline visualization lies in seed placement, i.e., where to drop seeds and how many seeds should be placed. Seeding too many or too few streamlines may not reveal flow features and patterns either because it easily leads to visual clutter in rendering or it conveys little information about the flow field. Not only does the number of streamlines placed matter, their spatial relationships also play a key role in understanding the flow field. Therefore, effective flow visualization requires the streamlines to be placed in the right place and in the right amount. This paper introduces hierarchical streamline bundles, a novel approach to simplifying and visualizing 3D flow fields defined on regular grids. By placing seeds and generating streamlines according to flow saliency, we produce a set of streamlines that captures important flow features near critical points without enforcing the dense seeding condition. We group spatially neighboring and geometrically similar streamlines to construct a hierarchy from which we extract streamline bundles at different levels of detail. Streamline bundles highlight multiscale flow features and patterns through clustered yet not cluttered display. This selective visualization strategy effectively reduces visual clutter while accentuating visual foci, and therefore is able to convey the desired insight into the flow data.

Index Terms—Streamline bundles, flow saliency, seed placement, hierarchical clustering, level-of-detail, flow visualization.

1 Introduction

Flow visualization is an important topic in scientific visualization. Researchers have developed various flow visualization techniques, such as feature extraction and tracking [27], dense and texture-based techniques [16], topology-based techniques [17], partition-based techniques [30], and integration-based techniques [24]. We refer interested readers to Weiskopf and Erlebacher [41] for an overview of flow visualization. This paper focuses on integration-based techniques, i.e., streamline visualization.

Verma et al. [37] proposed three criteria for effective streamline placement and visualization: *coverage*, *uniformity*, and *continuity*. While capturing important features and revealing flow continuity are essential for generating correct, complete, and pleasing visualization results, one can still produce meaningful visualizations by not covering the entire domain and not adhering to the principle of uniformity. For example, Li et al. [19] demonstrated an illustrative technique that succinctly depicts a 2D flow field using a minimum set of streamlines. For 3D flow fields, placing evenly-spaced streamlines that cover the entire domain would inevitably lead to visual clutter when projected to 2D for viewing. This poses a major obstacle for effective visual understanding. Therefore, a visualization that is concise but still captures critical flow features is highly desirable. For real and complex flow data, prioritizing flow features enables clear and controllable viewing. We thus conjecture that a suitable solution for 3D flow visualization is to selectively display streamlines that highlight important flow features at various levels of detail (LODs). This paper presents a technique that realizes this idea.

We present *hierarchical streamline bundles*, a new technique for summarizing and visualizing flow fields. Given an input flow field, our method first generates a set of streamlines that captures important flow features. The seeding is guided by the saliency map derived from the differences of Gaussian-weighted averages of curvature and torsion fields at multiple scales. We then cluster streamlines by grouping spatially neighboring and geometrically similar streamlines in a hierarchical manner. Hierarchical streamline bundles are created at different LODs from streamlines that are close to the boundaries of neighboring clusters. The saliency-guided seeding strategy allows us to purposefully capture prominent flow features without enforcing the dense seeding condition, thus it is more efficient than random or uniform seeding that does not consider flow characteristics. Furthermore, our seeding does not extract critical points explicitly as required in template-based seeding [37], [43]. In practice, this provides a viable alternative to capture flow features as critical points are often difficult to find in a robust manner. We note that the general idea of hierarchical streamline bundles can be applied to a set of streamlines produced from different seeding strategies. The construction of a streamline hierarchy allows us to produce multiscale streamline clusters from which streamlines lying on cluster boundaries are extracted. Together, these boundary streamlines at a certain LOD form the streamline bundles. For saliency-guided seeding, hierarchical streamline bundles organize representative streamlines in the coarse-to-fine manner, which makes it ideal for prioritizing a large, complex flow field to enable flexible multiscale flow feature exploration and observation. Such a capability is clearly more desirable than overloading the viewers by presenting all flow features, large or small, simultaneously.

Our work is inspired by fiber clustering in diffusion tensor

• *H. Yu and J. H. Chen are with the Combustion Research Facility, Sandia National Laboratories, Livermore, CA 94550. Email: {hyu, jhchen}@sandia.gov.*

• *C. Wang and C.-K. Shene are with the Department of Computer Science, Michigan Technological University, Houghton, MI 49931. Email: {chaoliw, shene}@mtu.edu.*

imaging (DTI) visualization [26] and edge bundling in tree and graph visualization [9]. Clustering neighboring fibers traced from DTI data allows clear observation of the fiber structure and patterns. For tree and graph data, creating bundles from adjacency edges significantly increases the readability of the tree or graph being visualized. Both methods share the common theme of reducing visual clutter and facilitating data understanding. In our scenario, streamline bundles extracted are able to capture prominent flow features in a visually-striking way. Note that edge bundling determines both edge grouping and the paths of the edges. Our streamline bundles, like fiber clustering, only determine grouping. Streamline bundles have the following major advantages. First, streamline bundles are guaranteed to pass through the vicinity of critical points and are thus representative. Second, streamline bundles not only enforce visual clarity by selective display but also accentuate visual foci by clustered display. Third, streamline bundles are organized hierarchically and can capture flow features of varying scales at different LODs. Finally, streamline bundles form a partition of the underlying flow field and streamlines in each region of the partition are similar.

2 Related Work

2.1 Streamline Seeding and Visualization

Streamline seeding is important because it can greatly affect the quality of visualization. Simply placing seeds uniformly or randomly may not produce satisfactory results. As such, many research efforts have been devoted to develop good seeding strategies. Examples include image-guided streamline placement in 2D [36] and 3D [20], evenly-spaced streamline placement in 2D [14], [21] and on surfaces [33], the farthest point seeding strategy [25], the flow-guided method in 2D [37] and its extension to 3D [43], and dual streamline seeding [29]. Research efforts that are closely related to ours are priority streamlines [32], similarity-guided streamline placement [4], and illustrative streamline placement [19]. In our work, we trace streamlines by taking a saliency-guided seeding strategy that favors regions near critical points to capture important flow features. Our work thus also shares some similarity with the recent streamline selection work [23] in terms of capturing and highlighting flow features through selected streamlines.

2.2 Clustering and Bundling

Clustering large and complex flow fields promotes the understanding of flow structure. Clustering can be operated on the original vector data [8], [34] or the integral line data [26]. To cluster vector data, we can take a top-down [8] or bottom-up [34] approach. Clustering integral lines is commonly found in visualizing DTI data [26] where individual fibers are reconstructed and clustered to obtain bundles for easy understanding. The spatial proximity between two fibers indicates their similarity and thus can be used in fiber clustering [44]. Different distance measures have been proposed, including the average of point-by-point distances between corresponding pairs, the mean of closest point distances [5], the thresholded average distance [3], the weighted normalized sum of minimum distance [13], and the distance measured in a transformed feature space. We advocate streamline clustering instead of vector clustering since streamlines traced over the domain constitute continuous regional or global patterns rather than local entities of individual vectors. To achieve continuity, we favor long streamlines and do not impose constraints such as the separating distance between streamlines. Other advanced techniques that share the same spirit with our streamline bundles include multiscale flow field decomposition using algebraic multigrids [7], streamline predicates for flow structure definition [31], and pathline clustering for solvent molecules based on similarity evaluation of dynamic properties [2]. Compared with these techniques, our work is more intuitive: it is simpler in concept and easier to understand.

2.3 Curvature-Guided Visualization

In graphics and visualization, the gradient derived from the field has been widely utilized in tasks such as shading calculation and transfer function specification. Besides this first-order derivative, second-order derivatives such as the curvature have also been leveraged [10], [22], [15]. We utilize the curvature information derived from the flow field to prioritize the seeding so that important flow features and patterns can be captured in the streamline representation. Unlike previous work, we consider curve-based, not surface-based curvatures. The curvature information can be computed directly from the flow field without knowing the tangent curve itself.

3 Our Approach

Figure 1 sketches the major steps of our approach to generate hierarchical streamline bundles. In the following, we describe these steps in detail.

3.1 Streamline Generation

3.1.1 Flow Saliency

Our streamline seeding is guided by the saliency of a flow field. In general, the *saliency* of an item, e.g., a voxel in the flow field, is its state or quality of standing out with respect to neighboring items. For example, in 2D images, color, intensity, and orientation are the most important attributes for saliency detection [11]. For 3D meshes, geometric shapes such as the changes in the curvature lead to the notion of mesh saliency [18]. A saliency map can be computed from the data using the "center-surround" operations [11].

We observe that in 3D, the most characteristic properties of streamlines are their *curvature* and *torsion* as these two quantities capture the intrinsic nature of each individual streamline in space, thus yielding information about the behavior of the entire flow field. Therefore, we propose to compute the saliency of a flow field based on the Gaussian-weighted center-surrounding evaluation of the curvature and torsion values. The seeding is then prioritized according to flow saliency such that important characteristics can be captured.

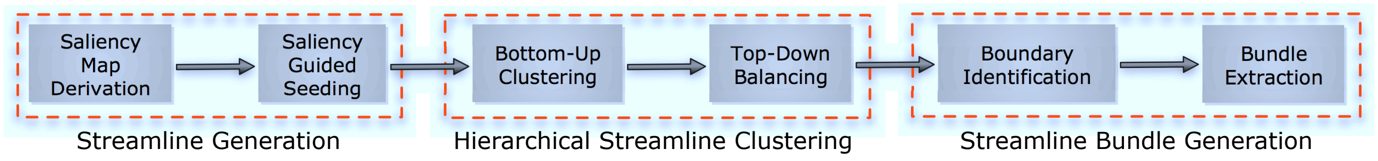


Fig. 1. The major steps of our approach to generate hierarchical streamline bundles.

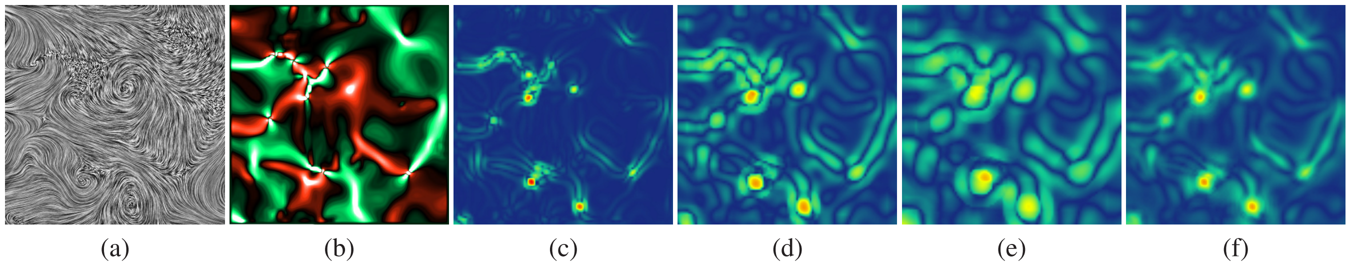


Fig. 2. Flow saliency. (a) is a LIC image of the 2D flow field of an earthquake simulation data set. (b) shows the curvature field (green for negative value, red for positive value, black for zero, and white for infinity). We compute the saliency maps at five scales (2ε to 6ε) where ε is 0.7% of the image's diagonal. (c)-(e) show three selected scales of 2ε, 4ε, and 6ε, respectively. The saliency increases from blue to green to yellow to red. (f) shows the final saliency map that combines all five saliency maps with a non-linear normalization. We use the final saliency map to guide the seeding.

3.1.2 Curvature and Torsion

In general, there are two solutions to compute curvature and torsion from an input vector field. One solution is to compute them from the tangent curves. In this case, explicit numerical integration along the vector field is required. A more efficient solution is to compute the curvature and torsion directly from the vector field and its partial derivatives [40]. Given a 3D vector field $V = (u, v, w)^T$, let $L(t)$ be an arbitrary tangent curve of V and let $P \in L$ be an arbitrary point on L. Furthermore, let $L(t)$ be parameterized in such a way that $P = L(t_0)$ and $\dot{L}(t_0) = V(L(t_0))$. We can obtain the n-th derivative vector of L $(n > 1)$:

$$\overset{n}{L}(t_0) = (u \cdot \overset{n-1}{L_x} + v \cdot \overset{n-1}{L_y} + w \cdot \overset{n-1}{L_z})(P), \tag{1}$$

where $\overset{1}{L} \mathrel{\widehat{=}} \dot{L}$, $\overset{2}{L} \mathrel{\widehat{=}} \ddot{L}$, and $\overset{3}{L} \mathrel{\widehat{=}} \dddot{L}$ etc.

The curvature measures how quickly a curve changes its unit tangent with respect to its arc length. In 2D, the signed curvature of point $P = L(t_0)$ is computed as [35]:

$$\kappa(t_0) = \frac{\det[\dot{L}(t_0), \ddot{L}(t_0)]}{\| \dot{L}(t_0) \|^3}. \tag{2}$$

Curves in 3D always have non-negative curvatures. The curvature of 3D point $P = L(t_0)$ is computed as [6]:

$$\kappa(t_0) = \frac{\| \dot{L}(t_0) \times \ddot{L}(t_0) \|}{\| \dot{L}(t_0) \|^3}. \tag{3}$$

The torsion measures how sharply a curve twists out of its osculating plane. The torsion of 3D point $P = L(t_0)$ is computed as [6]:

$$\tau(t_0) = \frac{\det[\dot{L}(t_0), \ddot{L}(t_0), \dddot{L}(t_0)]}{\| \dot{L}(t_0) \times \ddot{L}(t_0) \|^2}. \tag{4}$$

Since there is one and only one tangent curve through every non-critical point of the vector field V, we can define two scalar fields for V: the curvature field $\kappa(V)$ and the torsion field $\tau(V)$. Note that $\kappa(V)$ and $\tau(V)$ are not defined at critical points. $\tau(V)$ is also not defined at points where $\kappa = 0$.

In Figure 2 (a) and (b), we show an example of a 2D velocity field of an earthquake simulation data set and its curvature field. At critical points, the denominator of Equation 2 is zero. We thus assign the maximum curvature value computed from all the other points to $\kappa(V)$. Theisel and Rauschenbach [35] showed that for non-degenerate critical points, a highlight in the curvature visualization always indicates a critical point in the flow field and vice versa. From Figure 2 (b), we can observe that critical points in the flow field are captured in the curvature visualization.

3.1.3 Saliency Map

A straightforward metric that only considers the values of curvature and torsion may not be able to capture the perceptual importance of a flow field. For instance, repeated patterns, even if high in curvature or torsion, are visually uniform. It is the saliency that enables locations which stand out from their surrounding to persist. This center-surround idea can be implemented as the difference between the Gaussian-weighted averages computed at fine and coarse scales [11], [18]. Let $\mathscr{G}(|\kappa(P)|, \sigma)$ denote the Gaussian-weighted average of the *absolute* curvature at point P where σ is the threshold distance that we consider as neighboring points of P. We define the curvature saliency of P at a scale level i as:

$$S_{\kappa(P),i} = |\mathscr{G}(|\kappa(P)|, \sigma_i) - \mathscr{G}(|\kappa(P)|, 2\sigma_i)|, \tag{5}$$

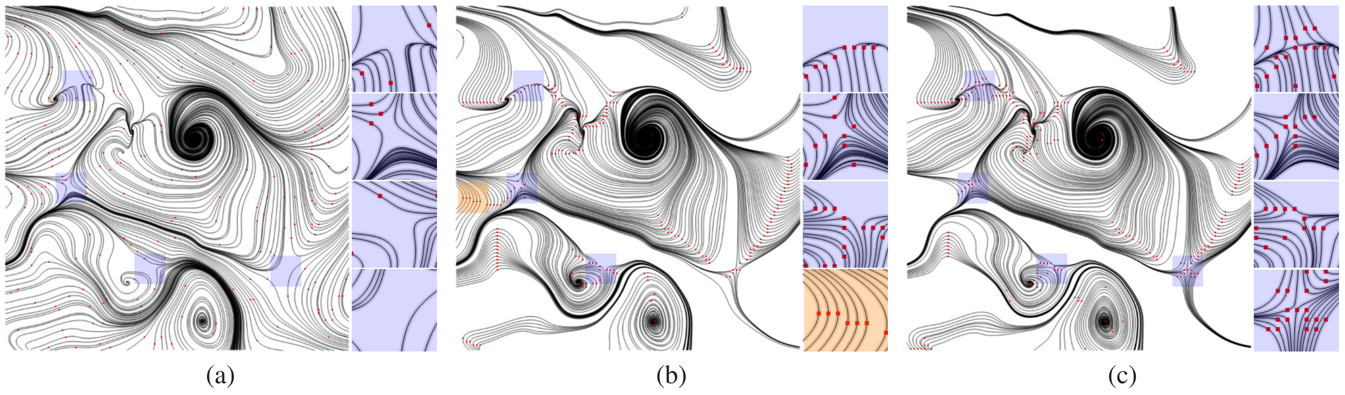


Fig. 3. Comparison of (a) random seeding, (b) curvature-guided seeding, and (c) saliency-guided seeding on the 2D flow field of an earthquake simulation. Each places 218 seeds (shown in red). Random seeding may miss important features (e.g., the four saddles shaded in light blue in (a)). Curvature-guided seeding may give a high priority to the regions that have high curvature values but are visually uniform (e.g., the region shaded in light orange in (b)), while leaving more important features only partially covered (e.g., the three saddles shaded in light blue in (b)). Saliency-guided seeding is able to place streamlines closer to critical points, ensuring that important features are well captured (e.g., the four saddles shaded in light blue in (c)). The resulting set of streamlines is more suitable for the subsequent bundle extraction and feature highlighting.

where σ_i is the standard deviation of the Gaussian filter at scale i. Similarly, we define the torsion saliency as:

$$S_{\tau(P),i} = |\mathscr{G}(|\tau(P)|,\sigma_i) - \mathscr{G}(|\tau(P)|,2\sigma_i)|. \tag{6}$$

In this paper, we use five scales $\sigma_i \in \{2\varepsilon, 3\varepsilon, 4\varepsilon, 5\varepsilon, 6\varepsilon\}$, where ε is a fixed percentage of the domain's diagonal. To combine the saliency maps $S_{\kappa(P),i}$ and $S_{\tau(P),i}$ at different scales and modalities, we utilize a nonlinear normalization operator $\mathscr{N}$ proposed by Itti et al. [11]. Globally, the operator promotes maps in which a small number of strong peaks of activity is present, while suppressing maps which contain numerous comparable peak responses. We apply these three steps to each saliency map: First, we normalize the values in the map to a fixed range $[0..M]$ to eliminate modality-dependent amplitude differences. Second, we find the location of the map's global maximum M and compute the average $\overline{m}$ of all its other local maxima. Finally, we multiply every point in the map by $(M - \overline{m})^2$. The final saliency map is computed as the summation of all normalized curvature and torsion saliency maps:

$$S_P = \frac{1}{2}\Big(\sum_i \mathscr{N}(S_{\kappa(P),i}) + \sum_i \mathscr{N}(S_{\tau(P),i})\Big). \tag{7}$$

Since planar curves have no torsion, the saliency map of a 2D flow field is simply the summation of all normalized curvature saliency maps, i.e., $S_P = \sum_i \mathscr{N}(S_{\kappa(P),i})$. Figure 2 (c)-(f) show the saliency maps at three different scales and the final saliency map for the given flow field.

3.1.4 Saliency-Guided Seeding

With the final saliency map, our seeding method selects seeds from locations in the order of decreasing saliency. To favor long streamlines, we integrate streamlines until they leave the domain or reach critical points. A large threshold is set for streamline length to avoid the looping case. If a seed is placed exactly on the critical point, then there is no streamline traced and this seed is discarded. If a candidate seed is on any of the streamlines previously placed (i.e., passing through a pixel/voxel belonging to a seeded streamline), then we discard this seed as well. The advantage of our saliency-guided seeding over random or uniform seeding is that it places seeds close to critical points and thus purposefully generates a set of candidate streamlines suitable for the following bundle extraction and feature highlighting. As a comparison, Figure 3 shows the streamlines generated using random seeding, curvature-based seeding, and saliency-based seeding, respectively, on a 2D flow field. Curvature-guided seeding uses absolute curvature values to prioritize candidate seeds. Unlike seeding directly using the curvature field, the saliency map is able to remove noise in the curvature field and allow streamlines to be placed closer to critical points. The seeding process can be terminated when high saliency regions corresponding to features have been covered.

3.2 Hierarchical Clustering

3.2.1 Similarity Measure

The output of our saliency-guided streamline generation is a set of streamlines F with each represented by a set of 3D points p_k, i.e., $F = \{F_i \mid F_i = \{p_k\}\}$. The similarity measure can be defined using the Euclidean distance between pairs of points on two input streamlines F_i and F_j. For example, we can form point pairs by mapping each point of one streamline to the *closest* point on the other streamline [5]. Three pairwise distances between F_i and F_j can be used: the closest point distance, mean of closest point distances, and Hausdorff distance. As suggested by Moberts et al. [26], we use the mean of closest point distances in our implementation, which is defined as:

$$d_M(F_i,F_j) = \frac{1}{2}\big(d_m(F_i,F_j) + d_m(F_j,F_i)\big), \tag{8}$$

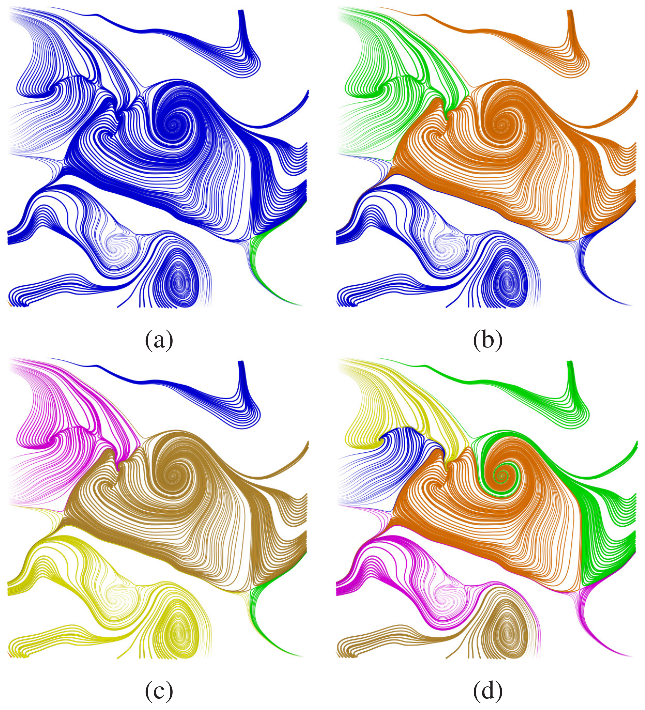


Fig. 4. (a) and (c) shows two coarse LOD results with three and six clusters, respectively, after the bottom-up clustering process. Each cluster uses a different color. Notice the outlier cluster with only one very short streamline at the very bottom-left corner in both images. (b) and (d) shows the corresponding LOD results after we perform the top-down balancing. We can observe that the top-down balancing takes care of the outliers in the streamline clusters and presents a more desirable output hierarchy.

where

$$d_m(F_i,F_j) = \frac{\sum_{p_k\in F_i} \min_{p_l\in F_j} \|p_k - p_l\|}{N},$$

and N is the number of points sampled along F_i. $d_m(F_j,F_i)$ is defined similarly.

3.2.2 Bottom-Up Clustering

To cluster the streamlines, we use an agglomerative hierarchical clustering. This bottom-up method begins with each streamline in a distinct cluster, and successively merges the two most similar clusters together until a stopping criterion is satisfied. Different variations of hierarchical clustering algorithms exist depending on how the similarity between a pair of clusters is defined. Among them, the single-link and complete-link algorithms are most popular ones [12]. In the *single-link* method, the distance between two clusters is the *minimum* of the distances between all pairs of items (one item from the first cluster and the other from the second). In the *complete-link* method, the *maximum* of the distances between all pairs of items is used. In either case, two clusters are merged to form a larger cluster based on minimum distance criteria. Hierarchical algorithms are more versatile than partitional algorithms such as the k-means algorithm. For example, the single-link clustering method performs well on data sets containing non-isotropic clusters, including well-separated, chain-like, and concentric clusters while the k-means clustering algorithm works well only on data sets having isotropic clusters.

Moberts et al. [26] evaluated the combinations of four different clustering methods (single-link, complete-link, weighted-average, and shared nearest neighbor) and four different similarity measures (closest point distance, mean of closest point distances, Hausdroff distance, and end point distance) in the context of DTI fiber clustering. They reported that the use of hierarchical single-link clustering combined with the mean of closest point distances gives the best results. Since DTI fibers and general streamlines share great similarity, we follow their suggestion and use the single-link with the mean of closest point distances in our streamline clustering.

During the clustering process, we generate a binary tree to indicate which two clusters are merged at each iteration. Each node in the tree has an attribute, *step*, recording at which step the associated cluster is created. By default, $step = 0$ for all leaves and $step = M-1$ for the root, where M is the number of input streamlines. For each tree node, we also assign an attribute, *size*, recording the number of streamlines in its cluster. $size = 1$ for all leaves and $size = M$ for the root.

3.2.3 Top-Down Balancing

An issue we experience with the hierarchical clustering is that the resulting binary tree may not be balanced. By *balanced*, we mean that for all nodes at a coarse LOD, their corresponding streamline numbers (i.e., their *size* values) should be similar. A balanced clustering tree leads to LOD refinement that is consistent with our intuition. The reason that we may produce an unbalanced clustering tree is because some streamlines may have very short paths traced over the domain. These streamlines are typically along the domain boundary or are between close critical points, which appear as outliers or noise [1]. They are merged very late in the bottom-up clustering process and stay close to the root. Thus, they will show up even though we select a few clusters for display, which is not desirable. Figure 4 (a) and (c) show two LODs of the earthquake data set to demonstrate the unbalanced streamline hierarchy. We can see that it is usually difficult for users to perceive such small outliers even at a coarse LOD. Especially for 3D vector fields, short streamlines can be easily occluded, and the corresponding browsing through LODs does not best match a user's general intuition on how an object should be disassembled.

A possible solution to overcome this problem is to traverse a clustering tree in a different way. For instance, in the field of software evolution, Voinea and Telea [39] proposed two methods to generate partitions with near same sizes or near internal similarities from a highly unbalanced clustering tree. However, with their pre-order tree traversal scheme, it is still possible to visit and split smaller tree nodes before visiting larger ones, provided that these smaller tree nodes are close to the root as shown in our study.

We perform a top-down balancing process after the bottom-up hierarchical construction. The pseudocode is given in Algorithms 1, 2, and 3. In BALANCETREE, we start with a LOD only consisting of the root of the binary tree and refine

Algorithm 1 BALANCETREE(*rootnode*: treenode)

```
// Create a new list of treenodes sorted in the order of decreasing size
$nodelist \leftarrow$ new multimap <int, treenode, greater<int>>
$nodelist.insert(rootnode.size, rootnode)$
$M \leftarrow$ the total number of input streamlines
$i \leftarrow 0$
while true do
    // Get the largest node in the list
    $activenode \leftarrow nodelist.begin()$
    // Stop the loop when nodelist only contains the leaves
    if activenode is a leaf then
        break
    end if
    // Compute clusternum if activenode is split into two
    $clusternum \leftarrow nodelist.size() + 1$
    // Compute the average size
    $avg \leftarrow root.size/clusternum$
    // Balance the subtree rooted at activenode if not all its children are balanced
    if $activenode.child_l.size/avg < \lambda_b$ or $activenode.child_r.size/avg < \lambda_b$ then
        BALANCESUBTREE(rootnode, activenode, clusternum)
    end if
    // Assign a new step value to activenode
    $activenode.step \leftarrow M - 1 - i$
    $i \leftarrow i + 1$
    $nodelist.insert(activenode.child_l.size, activenode.child_l)$
    $nodelist.insert(activenode.child_r.size, activenode.child_r)$
    $nodelist.erase(activenode)$
end while
```

Algorithm 2 BALANCESUBTREE(*rootnode*: treenode; *activenode*: treenode; *clusternum*: int)

```
$nodelist \leftarrow$ new multimap <int, treenode, greater<int>>
$nodelist.insert(activenode.child_l.size, activenode.child_l)$
$nodelist.insert(activenode.child_r.size, activenode.child_r)$
while true do
    $node \leftarrow nodelist.begin()$
    $avg \leftarrow root.size/clusternum$
    if $node.size/avg \geq \lambda_b$ and $node.sibling.size/node.size \geq \lambda_f$ then
        $seednode \leftarrow node.parent$
        break
    end if
    if node is a leaf then
        $seednode \leftarrow node.parent$
        break
    end if
    $nodelist.insert(node.child_l.size, node.child_l)$
    $nodelist.insert(node.child_r.size, node.child_r)$
    $nodelist.erase(node)$
    $clusternum \leftarrow clusternum + 1$
end while
LOCALCLUSTERING(rootnode, activenode, seednode)
```

the LOD by visiting the nodes in the decreasing order of their sizes. For each visited node, *activenode*, we examine whether its two children are balanced or not. For an unbalanced child, the ratio of its size to the average node size at the current LOD is less than λ_b. If the children are balanced, we continue our traversal process; otherwise, we perform the balancing process on *activenode* in BALANCESUBTREE. Then, for *activenode*, we assign a new step value, $step = M - 1 - i$, where M is the number of input streamlines and i is the step at which *activenode* is visited. We continue our traversal process until the entire tree is balanced.

In BALANCESUBTREE, we visit the descendants of *activenode* in the decreasing order of their sizes to find a node, *seednode*. There are two cases for *seednode*. In the first case,

Algorithm 3 LOCALCLUSTERING(*rootnode*: treenode; *activenode*: treenode; *seednode*: treenode)

```
$nodearray \leftarrow$ new array
$nodearray.add(seednode.child_l)$
$nodearray.add(seednode.child_r)$
// Gather the outliers
$node \leftarrow seednode$
while $node \neq activenode$ do
    $nodearray.add(node.sibling)$
    $node \leftarrow node.parent$
end while
Construct a new clustering tree of all the items in nodearray using the same bottom-up clustering method, where the distance between any two clusters containing $seednode.child_l$ and $seednode.child_r$ is set to $+\infty$.
// Update rootnode (if needed), and activenode
if $rootnode = activenode$ then
    $rootnode \leftarrow root$ of the new tree
end if
$activenode \leftarrow root$ of the new tree
```

the ratio of one *seednode*'s child size to the average node size is greater than λ_b, and the ratio between two children's sizes is greater than λ_f. The effects of λ_b and λ_f will be discussed in Section 4.4. In the second case, *seednode*'s children are two leaves, which means that the tree rooted at *activenode* is close to a vine (i.e., a chain-shape degenerate binary tree that is highly unbalanced). We can see that in both cases the sibling of each node along the path from *seednode* to *activenode* is corresponding to a small cluster (i.e., an outlier).

In LOCALCLUSTERING, we gather these outliers as well as *seednode*'s children, and perform a localized bottom-up clustering within these nodes. The distance between any two clusters containing *seednode*'s children is set to $+\infty$. The distance between any other two clusters is still computed using the single-link with the mean of closest point distances. By this means, we obtain a refined subtree rooted at *activenode* that has two more balanced branches, and *seednode*'s children are separated into these two branches. Meanwhile, the original clustering result is largely preserved in the refined subtree. This is because we use the same similarity measurement during the localized clustering, and the subtree rooted at those outliers remains intact. As a result, we can ensure that the top-down balancing process leverages the bottom-up clustering results as much as possible for an efficient restructuring of the clustering tree. We present a quantitative balancing analysis in Section 4.4 to validate our statement. Figure 4 (b) and (d) show the LOD results after the top-down balancing process. As we can see, the balanced streamline hierarchy yields the LODs that are more consistent with our intuition.

3.3 Streamline Bundle Generation

3.3.1 Bundle Definition

Given a cluster in the hierarchy, there are two ways to define *representative* streamlines and form the bundle. We can either use the streamlines close to the cluster *centroid* or use the streamlines along the cluster *boundary*. In this paper, we define the *cluster boundary* as the closest neighboring streamlines that belong to different clusters. Choosing streamlines close to the centroid is commonly used in most cases. However, to form streamline bundles, we choose boundary streamlines to represent the cluster. Our rationale is that streamline bundles

Algorithm 4 COMPUTECONFIDENCE(*node*: treenode)

```
// Compute boundary cells
$bcells \leftarrow node.cells \cap node.sibling.cells$
// Identify boundary lines
if 2D grids then
    $pointlist \leftarrow$ new list
    for each cell $c$ in $bcells$ do
        for each line $l$ in $c.lines$ do
            Compute $l$'s intersections on $c$'s boundary and insert them into $pointlist$
        end for
        Sort $pointlist$ in the clockwise order
        Identify all pairs of consecutive intersections in $pointlist$ that belong to two boundary lines of different clusters, and insert the boundary lines of $node$ into $node.boundarylines$
    end for
else if 3D grids then
    for each cell $c$ in $bcells$ do
        $linelist \leftarrow c.lines \cap node.lines$
        if $linelist.size() < node.lines.size()/\lambda_m$ then
            Randomly choose $\delta_n$ lines from $linelist$ and insert them into $node.boundarylines$
        end if
    end for
end if
// Inherit parent's boundary lines
if $node$ is not the root then
    $node.boundarylines.insert(node.parent.boundarylines)$
end if
for each line $l$ in $node.lines$ do
    $l.d_b \leftarrow$ $l$'s minimal distance to all lines in $node.boundarylines$
end for
for each line $l$ in $node.lines$ do
    Normalize $l.d_b$
    // Compute confidence value with respect to boundary lines
    $l.C_b \leftarrow 1 - l.d_b$
end for
// Find the central line of $node$
for each line $l$ in $node.lines$ do
    $std \leftarrow$ standard deviation of $l$'s distances to all lines in $node.lines$
    if $minstd > std$ then
        $minstd \leftarrow std$
        $central \leftarrow l$
    end if
end for
for each line $l$ in $node.lines$ do
    $l.d_c \leftarrow$ $l$'s distance to $central$
end for
for each line $l$ in $node.lines$ do
    Normalize $l.d_c$
    // Compute confidence value with respect to the central line
    $l.C_c \leftarrow l.d_c$
    // Compute the average confidence value
    $l.C_a \leftarrow (l.C_b + l.C_c)/2$
end for
Sort $node.lines$ in the nondecreasing order based on $C_a$
if $node$ is not a leaf then
    COMPUTECONFIDENCE($node.child_l$)
    COMPUTECONFIDENCE($node.child_r$)
end if
```

should highlight flow features and patterns such as critical points. For a source or sink, it does not matter if we select close-to-centroid or boundary streamlines since either group of streamlines can approach arbitrarily close to the source or sink. However, only boundary streamlines are closest to a saddle and are thus best to reveal the saddle together with other boundary streamlines of neighboring clusters. We illustrate this idea with an example data set in Figure 5. Therefore, we define *streamline bundles* as the union of streamlines that are close to the boundaries of neighboring clusters. With the agglomerative clustering, we can create streamline bundles at various LODs.

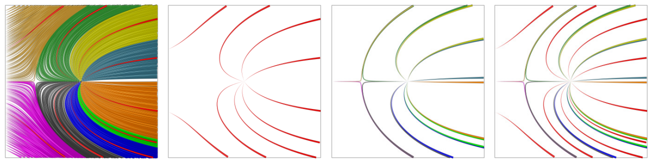


Fig. 5. Comparison of the effects between central and boundary streamlines of the clusters using a synthesized data set. (a) shows a certain LOD with 9 clusters, where red streamlines are the centroids of the clusters. (b) shows central streamlines only, where the saddle is not captured. (c) shows boundary streamlines only, where both the saddle and the sink are captured. (d) shows both central and boundary streamlines. Note that central streamlines in (d) will be captured by newer boundary streamlines added in (c) at a finer LOD.

3.3.2 Boundary Identification

Our boundary identification is sketched in Algorithm 4, which is performed in a top-down fashion. We first identify boundary streamlines based on the neighborhood relationship among the clusters. We take a grid, which has the same resolution as the original flow field, to cover the domain. For each tree node, *node*, it stores all its streamlines in the list *node.lines*, and stores all the grid cells intersecting with its streamlines in the list *node.cells*. For each cell c, it stores all its intersecting streamlines in the list $c.lines$.

For each node, we first obtain a set of its boundary grid cells that are shared with its sibling, and then identify the boundary streamlines within each boundary grid cell. For 2D grids, we compute the streamline intersections on the grid cell's boundary and use them to identify the boundary streamlines. For 3D grids, we opt for a simpler solution that randomly chooses the streamlines on the grid cell as boundary streamlines. The number of lines selected is constrained by a predefined threshold δ_n (which is a small integer such as four or five). However, when a grid cell either covers or is close to critical points, such as sinks or sources, it might intersect with most of the streamlines of its intersecting cluster with respect to a ratio λ_m (we set $\lambda_m = 0.5$). In this case, the randomly selected streamlines may not lie on the boundary and we just discard the grid cell from consideration. This process continues until all boundary cells have been checked. Moreover, each non-root node also nicely inherits the boundary streamlines identified from its parent node. In this way, we can identify a set of boundary streamlines for a cluster. For each streamline, we assign a confidence value with respect to the boundary streamlines, $C_b = 1 - d_b$, where d_b is the streamline's normalized minimal distance to all boundary streamlines. A streamline with a larger (smaller) C_b is close to the boundary (center) of the cluster.

Since our saliency-guided seeding strategy does not require dense seeding, there can be some boundary streamlines that are not close to any neighboring clusters. We identify these

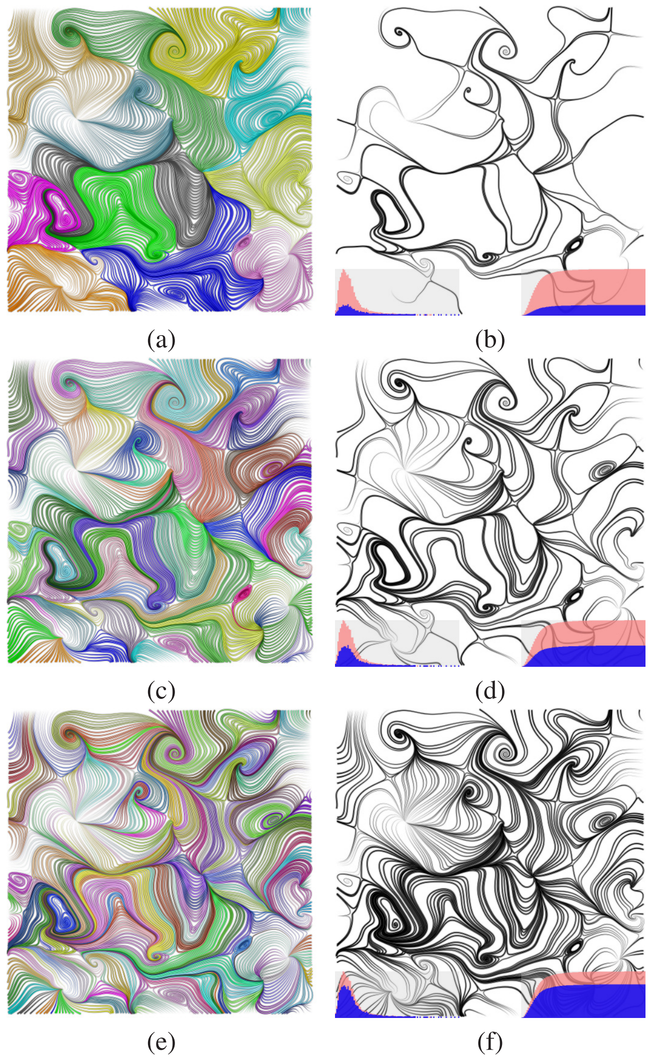


Fig. 6. (a), (c), and (e) show three LOD results with 12, 70, and 166 clusters, respectively, on another 2D earthquake flow field. (b), (d), and (f) show the streamlines that are close to the cluster boundaries. The saliency histogram and accumulated histogram are displayed in the lower left and right corners, respectively. Notice that streamline bundles identified in a lower resolution are part of bundles in a higher resolution, which makes the transition smooth between different LODs.

boundary streamlines based on the existing similarity characteristics. We first identify the *central* streamline of the cluster, i.e., the streamline whose standard deviation of the distances to all other streamlines in the cluster is minimal. Again, we use the mean of the closest point distances to compute the distance between two streamlines. For each streamline, we assign a confidence value with respect to the central streamline, $C_c = d_c$, where d_c is the streamline's normalized distance to the central streamline. A streamline with a larger (smaller) C_c is close to the boundary (center) of the cluster.

Finally, for a cluster, we assign an average confidence value, $C_a = (C_b + C_c)/2$, to each streamline in the cluster. We then sort all the streamlines in the nondecreasing order based on C_a. This order will be used to control the density of streamline bundles. Note that all the above steps for boundary identification are performed during the preprocessing stage.

3.3.3 Bundle Extraction

At runtime, we form the streamline bundle by first removing the central streamline, then iteratively removing the streamlines in the sorted order. The number of streamlines left can be controlled either with a certain percentage or a fixed number. Given a LOD, the user can interactively change either parameter and observe an animated effect illustrating how streamlines are removed successively from each cluster to reveal the bundles.

3.3.4 LOD Control

At runtime, the user can control the LOD by adjusting two parameters: the number of clusters c and the density of boundary streamlines ρ. Selecting a larger value for c reveals more details. The LOD selection is automatically determined by the step order in the binary clustering tree. ρ can be controlled via a threshold that determines how many percent of streamlines or how many streamlines should be kept to form the bundles. A larger value for the threshold leads to denser streamline bundles. Besides automatic LOD selection, the user may want to explore a certain region of interest. As such, we also allow the user to manually select a cluster and refine it to observe more details. In this way, the user can dissect the flow using a coarse LOD and selectively refine clusters of interest while keeping the rest of LOD as the context. This will produce a customized, multiscale summarization of the flow field to express the user's interest.

4 Results and Discussion

4.1 Data Sets and Timing Performance

We experimented our approach with six flow data sets. Two earthquake data sets are from the extracted slides of a 3D simulation for modeling ground motion of the 1994 Northridge earthquake. The hurricane data set is from a simulation of the Hurricane Isabel, a strong hurricane in the west Atlantic region in September 2003. The combustion data set is from a turbulent combustion simulation performed at the Sandia National Laboratories. The plume data set is from a simulation of solar plume at the National Center for Atmospheric Research. Finally, the supernova data set is from a simulation of the development of a rotational instability in a supernova shockwave. Table 1 lists these data sets, the number of streamlines we placed, the average number of points per streamline, and the timing for every step of our algorithm. Compared with the other three 3D flow data, the reason that the combustion data set has a much smaller average number of points per streamline is due to its turbulent nature (with the present of many small-scale critical regions) and the thin slab we took (20 voxel-wide along the z axis).

We used a hybrid CPU-GPU solution in our computation with the following hardware configuration: a quad-core Intel i7-975 3.33GHz processor with 6GB memory, and an nVidia GeForce GTX 285 with 1024MB video memory. All the

TABLE 1
The configurations of the flow data sets in our experiments, and their preprocessing timing performance in seconds.

data set	dimension	# streamlines	# avg points per streamline	saliency map (GPU)	streamline generation (CPU)	similarity measure (GPU)	bottom-up clustering (CPU)	top-down balancing (CPU)	boundary identification (CPU)	total preprocessing
earthquake (Fig. 2)	100×100	218	202	0.0011s	0.85s	0.09s	0.05s	0.01s	0.22s	1.22s
earthquake (Fig. 6)	100×100	826	334	0.0011s	23.25s	9.73s	1.22s	0.04s	1.85s	36.09s
hurricane (Fig. 7)	$500 \times 500 \times 100$	3000	828	2.4569s	116.54s	224.69s	14.42s	0.64s	1.34s	360.09s
combustion (Fig. 8)	$506 \times 400 \times 20$	3000	173	0.3712s	15.02s	31.26s	28.15s	4.17s	1.02s	79.99s
plume (Fig. 9)	$126 \times 126 \times 512$	2000	1495	0.7838s	279.98s	319.50s	11.09s	3.54s	1.46s	616.35s
supernova (Fig. 12)	$216 \times 216 \times 216$	3000	617	0.9965s	65.88s	141.57s	15.01s	1.01s	1.71s	226.18s

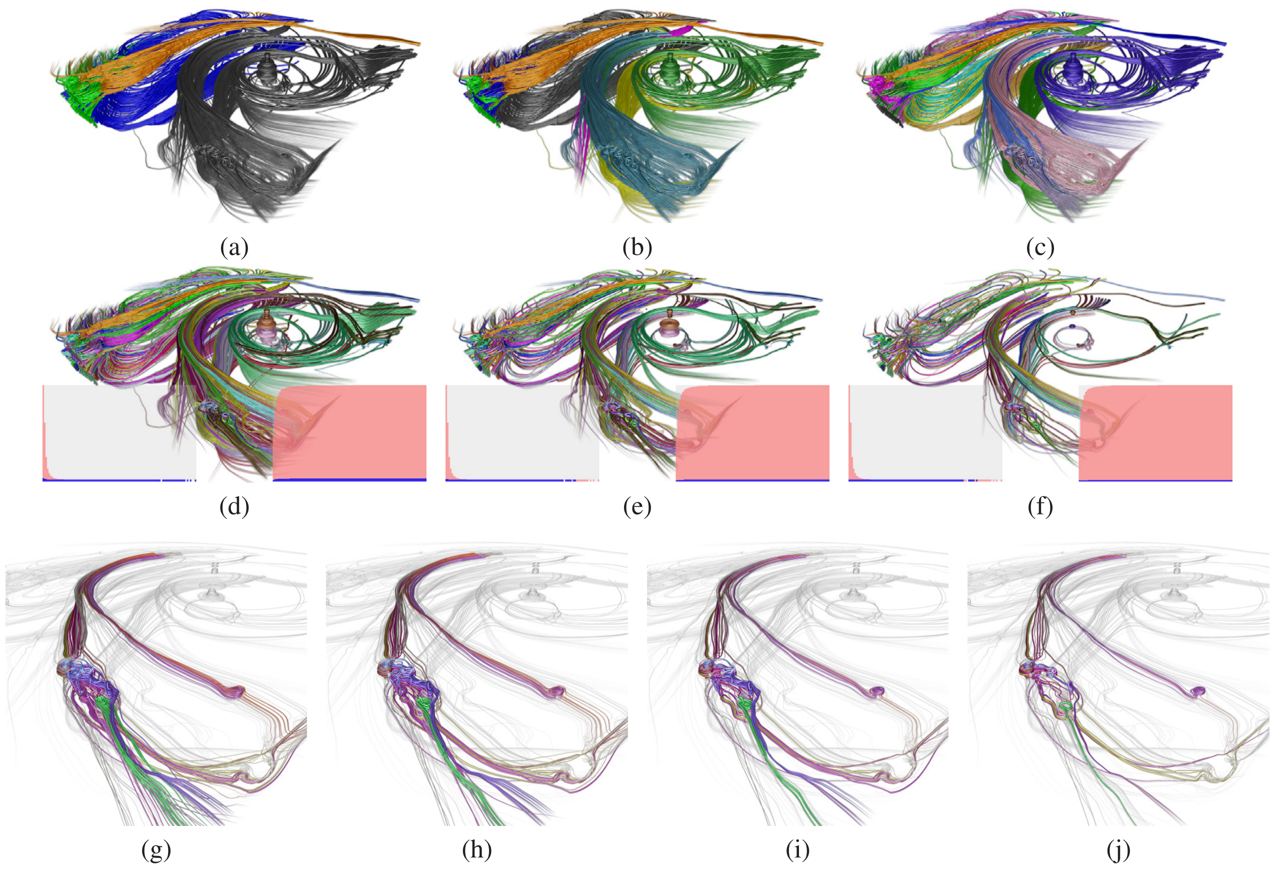


Fig. 7. (a)-(c) show the LOD refinement of the streamline hierarchy of a hurricane data set with 4, 8, and 18 clusters, respectively. (d)-(f) show the gradual removal of streamlines from 128 clusters to reveal the streamline bundles. (d) shows all streamlines, (e) shows 33% of streamlines in each cluster, and (f) only shows streamlines that are the cluster boundaries. The saliency histogram and accumulated histogram are displayed in the lower left and right corners, respectively. (g)-(j) show the user-specified clusters (in their original colors) as the focus and the rest (in gray) as the context. (g) shows all streamlines, (h) shows 66% of streamlines in each cluster, (i) shows 33% of streamlines in each cluster, and (j) only shows streamlines that are the cluster boundaries.

timing reported in Table 1 was for the steps conducted during the preprocessing. The computation of saliency map and similarity measure was performed using CUDA. The timing mainly reflects the complexity of each step. The complexity of saliency map computation is $O(kn)$, where n is the number of voxels and k is the size of Gaussian kernel, which could be costly to compute using the CPU for large n and k. For instance, it took about one hour to calculate the saliency map for the hurricane data set using the CPU, while it only took about two seconds using the GPU. The complexity of similarity measure computation is $O(m^2n^2)$, where n is the number of streamlines and m is the number of points per streamline. Computing this step using a single CPU could be prohibitively expensive. There are certain optimization approaches available. We used a CUDA implementation to accelerate the computation because the task is embarrassingly parallel, where computing the distance between each pair of streamlines can be independently performed by a CUDA thread. The bottom-up clustering takes $O(n^2 \log n)$ time, where n is the number of streamlines. This computation is affordable using the CPU for small n in the current setting. For boundary identification, the time spent on computing the confidence

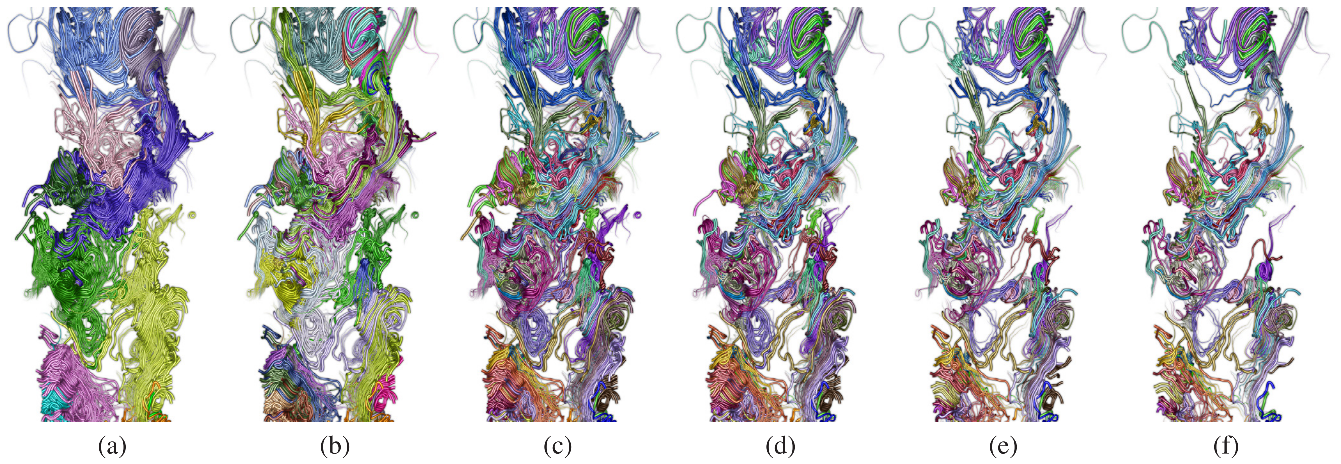


Fig. 8. A zoom-in into a turbulent combustion data set. (a)-(c) show the LOD refinement of the streamline hierarchy with 20, 100, and 200 clusters, respectively. (d)-(f) show the gradual removal of streamlines from 200 clusters to reveal the streamline bundles. (d) shows 66% of streamlines in each cluster, (e) shows 33% of streamlines in each cluster, and (f) only shows streamlines that are the cluster boundaries.

values is proportional to the number of streamlines, while evoking the calculations for boundary streamline identification largely depends on the closeness between the clusters which can vary among different data sets. At runtime, the bundle extraction time was less than 1ms for all test data sets, which was omitted in Table 1. The streamline hierarchy built in the preprocessing stage enables the user to control the LOD and explore the data interactively in real time. In the rendering, we use the tapering effect (i.e., varying the thickness along the streamlines) to highlight the flow direction.

4.2 2D Flow Field Results

Figures 2, 3, and 4 illustrate the process of our approach on a 2D earthquake flow field: deriving flow saliency, seeding based on the saliency, constructing the streamline hierarchy, and performing the balancing. Examples of LOD selection and streamline bundles on a more complex 2D earthquake flow field are given in Figure 6. As we can see, different LODs and their corresponding streamline bundles are able to capture essential flow features and structure in an adaptive manner. We also plot the saliency histograms and accumulated histograms in Figure 6 (b), (d), and (f). In both saliency and accumulated histograms, the horizontal axis represents the range of saliency values. The vertical axis of a saliency histogram represents the number of voxels with values equal to a given value, while the one of an accumulated histogram represents the number of voxels with values less than or equal to a given value. The histograms in pink are computed from the saliency values of all the voxels in the original field, and the ones in blue are computed from the saliency values of the voxels covered by the displayed streamlines. We can see that our streamline bundle representation is quite effective as high-saliency regions are well covered even though a smaller number of streamlines is chosen. Our solution is also quite efficient since we do not spend much ink (i.e., effective pixels) for drawing the 2D vector field, especially for Figure 6 (b) and (d), as evidenced by the accumulated histogram. Furthermore, our method guarantees that streamline bundles extracted in a lower resolution are part of the bundles in a higher resolution. This capability provides us a nice way to simplify the underlying flow field in multiple scales and produces a smooth transition between different LODs.

4.3 3D Flow Field Results

Figure 7 shows the results with a hurricane simulation data set. The first row shows the coarse-to-fine cluster selection. The second row shows the density control to reveal streamline bundles. As we favor long streamlines, many interesting small-scale features besides the hurricane's eye can be captured in streamline visualization. Such results are not available in other flow visualization work using the same data set. Moreover, notice how well Figure 7 (f) maintains detail flow features presented in (d). Streamline bundles succinctly depict the flow patterns using a small number of lines while in the meantime, accentuating visual foci via organizing them in the form of bundles. This is confirmed by the saliency histogram and accumulated histograms we plot: the accumulated histograms show that the streamlines only cover a small portion of the field, while the saliency histograms show that the streamlines capture the most salient regions. The third row shows focus+context visualization of streamline bundles. This happens when the user wants to focus on a few streamline clusters and explore them separately from the rest of clusters. We display clusters that are not selected in less detail and fade them into the context using an appropriate coloring scheme. The clusters of focus are thus standing out. The user can change the density for each cluster in focus for viewing. As we can see, Figure 7 (j) captures the overall and detail patterns shown in (g). This again confirms that our streamline hierarchy and bundling technique is able to preserve essential flow features with a small number of streamlines.

Besides the more structured hurricane flow, we also experimented with a very turbulent combustion flow, which is very challenging to visualize using existing streamline visualization

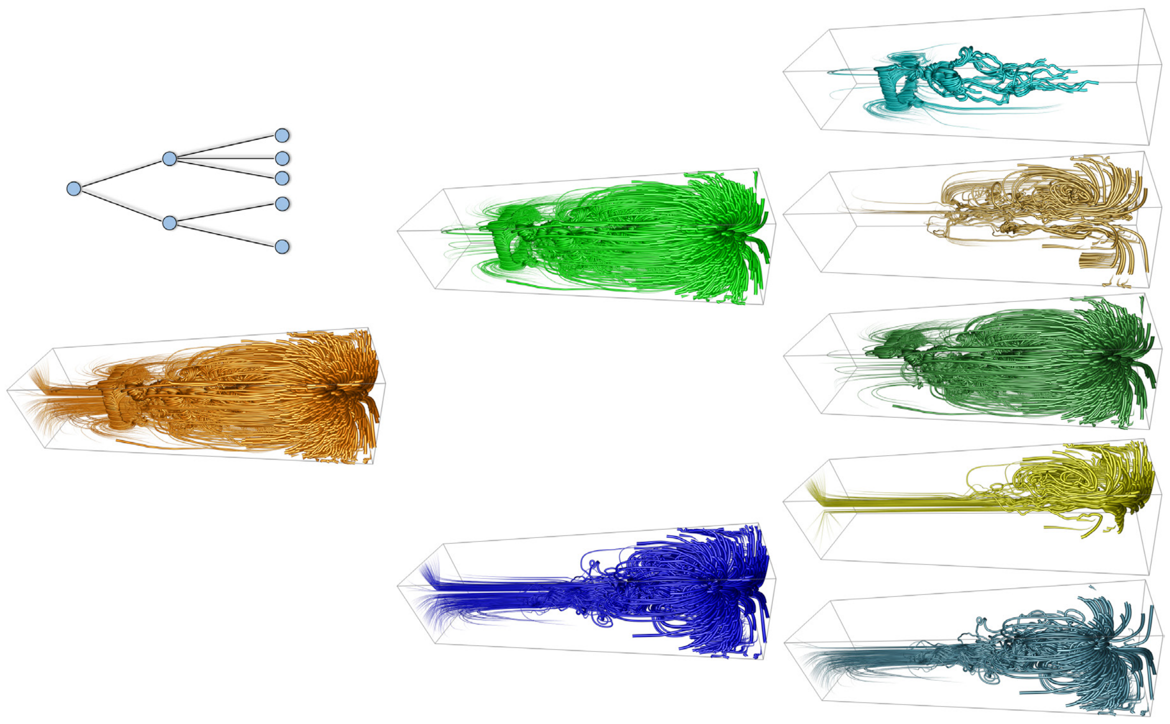


Fig. 9. Dissecting the plume data set. Leveraging the streamline hierarchy we have built, we can cut apart the underlying flow field to examine the internal structure in a coarse-to-fine manner. The explored hierarchy is illustrated in the top-left corner.

TABLE 2
The average standard deviation $\bar{s}$ of streamline numbers among clusters for different combinations of λ_b and λ_f.

λ_b \ λ_f	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.1	2.16	2.07	2.04	2.04	2.04	2.04	2.04	2.04	2.04	2.04
0.2	2.08	1.86	1.84	1.81	1.81	1.83	1.83	1.83	1.83	1.83
0.3	2.04	1.57	1.43	1.44	1.45	1.51	1.50	1.50	1.50	1.50
0.4	1.74	1.51	1.29	1.25	1.27	1.41	1.40	1.40	1.40	1.40
0.5	1.62	1.36	1.32	1.19	1.17	1.36	1.37	1.38	1.38	1.38

techniques. Specifically, we took a slab of dimension $506 \times 400 \times 20$ from the original field of dimension $506 \times 400 \times 100$ in our study. In Figure 8, we show streamlines partitioned into different numbers of clusters and the coarsening of streamlines in each cluster to reveal streamline bundles. We point out that although the underlying flow field is very turbulent, our approach could cluster streamlines into a hierarchy and still represent streamline bundles in a meaningful sense. Compared with Figure 8 (c) and (f), even though some small-scale features are lost in (f), much of the large-scale pattern remains.

The streamline hierarchy we build also allows us to partition the flow field. We can cut apart the field and observe individual parts separately and/or in the context. Figure 9 demonstrates an example with the plume data set. As occlusion is reduced, the entire set of streamlines can be viewed or understood better with the separation of different subsets of streamlines.

4.4 Quantitative Top-Down Balancing Analysis

We applied the *normalized information distance* (NID) proposed by Vinh et al. [38] as the metric to validate our top-down balancing method. As a general-purpose measure for comparing clusterings, NID has an advantage over the popular adjusted Rand index [28] in that it can be used in the non-adjusted form, thus enjoying the property of being a true metric in the space of clusterings. We used the 2D earthquake data set (Figure 3) with 218 streamlines and compared the quantitative results before and after top-down balancing. We selected different combinations of λ_b and λ_f for the balancing. After top-down balancing, for a given combination of λ_b and λ_f, we first calculated the standard deviation of streamline numbers among clusters at each LOD, i.e., s_i where $i = 2, \ldots, 218$ corresponding to the LODs with the cluster number ranging from 2 to 218. We then calculated the average of s_i as $\bar{s}$. Table 2 lists the values of $\bar{s}$ for the selected combinations of λ_b and λ_f. We can observe that for a given λ_b, the effect of changing λ_f is only marginal when λ_f is greater than 0.1. In addition, a larger value of λ_b corresponds to a smaller $\bar{s}$ since more clusters are involved in the balancing. Next, we validated our balancing method by fixing λ_f as 0.5 and choosing λ_b as 0.1, 0.2, 0.3, 0.4, and 0.5.

In Figure 10 (a), we compare the unbalanced clustering results to the balanced clustering results. Clearly, the results after balancing significantly reduce the difference of streamline numbers among the clusters when the number of clusters is small. Again, we can see that a larger λ_b gives a smaller standard deviation. As the standard deviation converges with

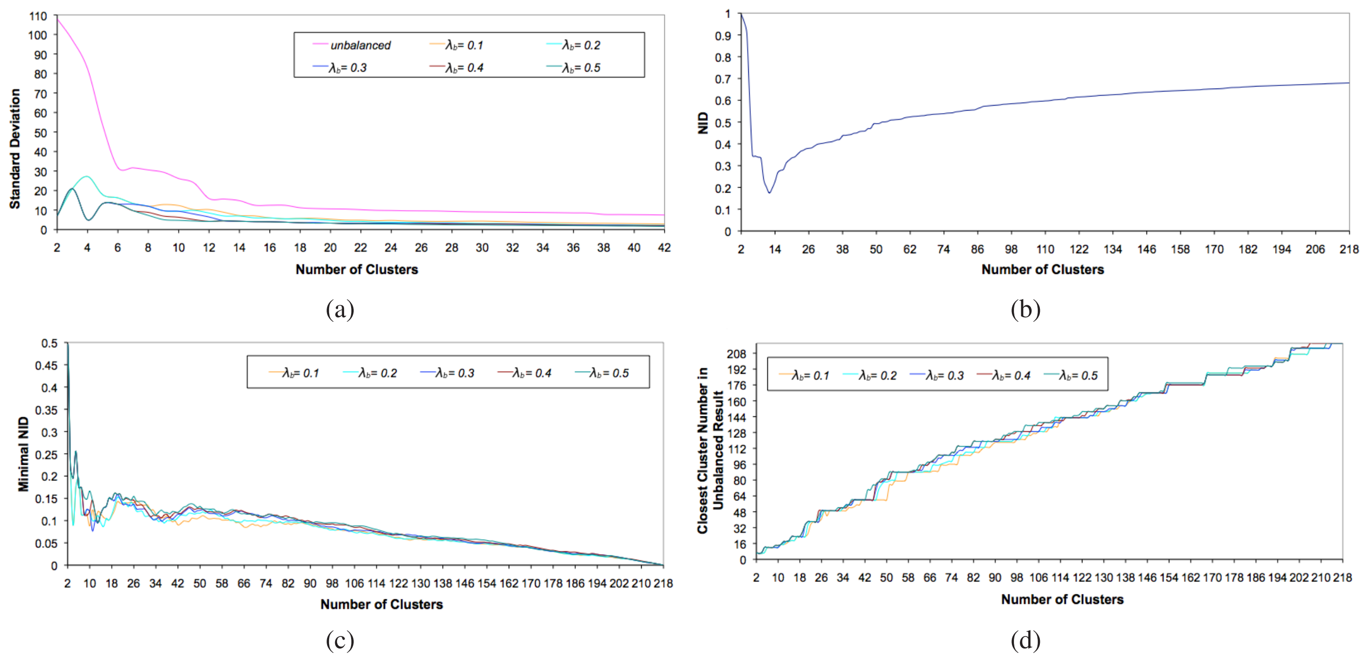


Fig. 10. Quantitative evaluation of top-down balancing using the earthquake data set. (a) is the comparison of the standard deviation of streamline numbers among clusters for the unbalanced case and the balanced cases with different parameter values. (b) shows the matching between the balanced result of 6 clusters with the unbalanced results. In (c) and (d), we plot for each cluster number in the balanced clustering results, the minimal NID value and the corresponding most similar cluster number from the unbalanced clustering results, respectively.

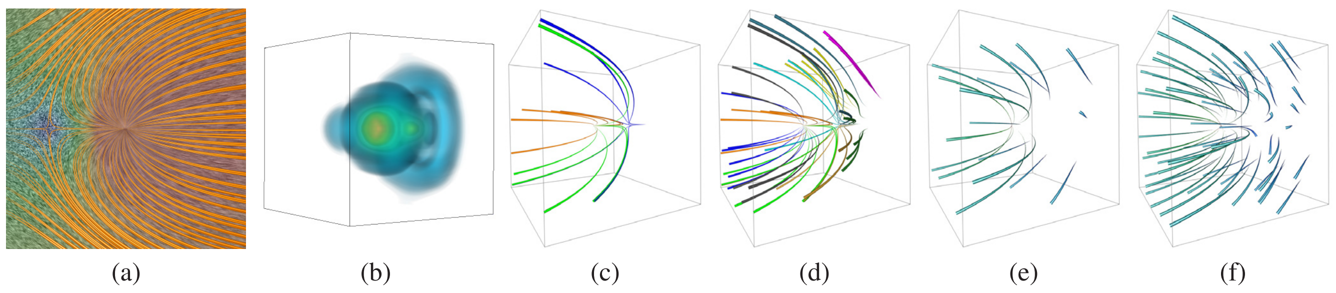


Fig. 11. Comparison of our approach with the vector field clustering method [34] using a synthesized data set that mimics the one used in their paper. (a) is a 2D slice view of the vector field. (b) is the 3D saliency field. (c) and (d) are our results with 3 and 10 clusters, respectively. (e) and (f) are the results using the previous method [34] with 18 and 60 clusters, respectively. Both (c) and (e) show 18 streamlines, and both (d) and (f) show 60 streamlines.

the increase of the number of clusters, we only show the results with the number of clusters ranging from 2 to 42 in the figure. In Figure 10 (b), we choose the clustering result of 6 clusters after the balancing where $\lambda_b = 0.3$ and measure the NID of this clustering result with the unbalanced clustering results (where the number of clusters ranges from 2 to 218). We can see that the balanced result with 6 clusters is most similar to the result of 12 clusters in the unbalanced case, with the minimum NID value of 0.174. This result indicates that our balancing method can effectively balance the streamline hierarchy, particularly close to the root where the original hierarchy is least balanced.

On the other hand, we point out that our balancing process does not dramatically alter the original streamline hierarchy. To confirm this, for each cluster number in our balanced clustering results, we found the minimal NID value (Figure 10 (c)) and the corresponding most similar cluster number (Figure 10 (d)) from the unbalanced clustering results. As we can see, for the clustering result at each level of our balancing method, we can always find a very similar cluster from the unbalanced clustering with a small NID value. In general, for the pair of clusters with the minimal NID value, the level in the unbalanced clustering is larger than the level in the balanced clustering. This is because the small clusters (i.e., outliers), which are close to the root in the unbalanced results, are moved down the hierarchy through the balancing. Our balancing method preserves the overall clustering results with the small NID values (shown in Figure 10 (c)) and achieves more balanced results with smaller standard deviation values (shown in Figure 10 (a)). The same conclusions can be drawn from the results with other data sets we experimented with.

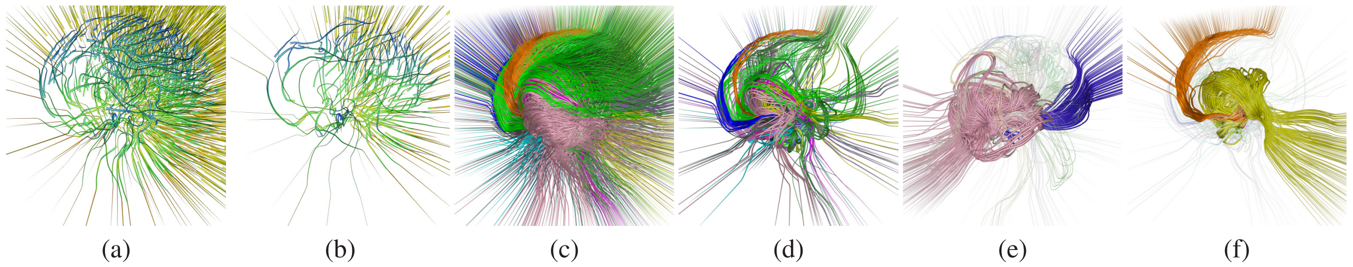


Fig. 12. Comparison of our approach with the vector field clustering method [34] using a supernova data set. (a) and (b) are the results using the previous method [34] with 3000 and 318 clusters, respectively. Our results with 24 clusters are shown in (c) and (d) with all streamlines and streamline bundles, respectively. (e) and (f) show the user-specified clusters as the focus and the rest as the context. Both (a) and (c) show 3000 streamlines, and both (b) and (d) show 318 streamlines.

4.5 Comparison with Vector Field Clustering

Notice that our saliency-guided seeding results are not the final results. They are only immediate results for deriving the streamline hierarchy and extracting streamline bundles. As such, instead of comparing with other seeding techniques, we compare our approach with the vector field clustering method presented by Telea and van Wijk [34]. Both solutions share the similarity of creating a hierarchy to simplify the flow field in an adaptive manner. While they cluster the original vector data, we cluster the traced streamlines.

Figure 11 shows the comparison using a synthesized vector field. From (a), we can clearly see that there are one sink and one saddle presented in the data. (b) shows a volume rendering of the saliency field, where the sink and saddle regions are with high saliency values. (c) and (d) show our results, where the streamlines are the cluster boundaries from the LOD refinement of the streamline hierarchy with 3 and 10 clusters, respectively, and the streamlines are colored with respect to the partitions. (e) and (f) show the results using Telea and van Wijk's method [34], where the streamlines are the representative streamlines from the vector field hierarchy with 18 and 60 clusters, and the streamlines are colored with respect to the velocity magnitudes. With our saliency-guided seeding and streamline bundle method, we can use a fewer number of streamlines to effectively capture the sink and saddle in this data.

We further conduct another experiment using the supernova data set. Figure 12 (a) and (b) show the results using Telea and van Wijk's method [34], where the streamlines are the representative streamlines from the vector field hierarchy with 3000 and 318 clusters, respectively, and the streamlines are colored with respect to the velocity magnitudes. (c) and (d) show our results, where the streamline bundles are revealed from 24 clusters, and the streamlines are color with respect to the partitions. (c) shows all streamlines, and (d) shows 10% of streamlines in each cluster. With our method, we can effectively place streamlines to cover important areas in the data and perform focus+context cluster highlighting. In (e) and (f), we can clearly see that two flow patterns moving along the opposite directions are identified and separated with streamline bundles. Our results help the scientists observe details of the flow field to verify their hypotheses.

4.6 Saliency and Critical Points

Utilizing the concept of flow saliency, we promote *critical points* to *salient regions* (refer to Figure 2 (b) and (h)). This makes it easier for us to capture flow features around critical points through saliency-guided seeding. The saliency tells how well locations standing out from their surround. In case that a flow field consists of uniformly-distributed repetitive critical points, then the non-linear normalization will suppress these small-scale features in the final saliency map. For complex 3D flows, this is actually desirable as we want to set higher priority to more salient critical regions than less salient critical points. This principle is also consistent with what Xu et al. [42] advocated in their recent work on entropy-based streamline seeding. While they had to perform thresholding or denoising to remove certain local maxima in the derived entropy field [42], our concept of flow saliency naturally takes care of this issue. In our current implementation, the seeding termination threshold is adjusted on a trial-and-error basis as the decision for high saliency value can vary among different data sets. A solution that automates this process is desirable.

4.7 Coverage, Uniformity, and Continuity

In terms of coverage, our method samples the "feature space" densely enough such that relevant features are reflected in streamline visualization. The spatial space, however, is selectively sampled to focus only on interesting flow patterns. Uninteresting regions that are not covered are more or less uniform. They can be safely inferred from streamlines surrounding them. In terms of uniformity, we do not maintain a uniform distribution of streamlines over the field. Instead, we keep high density along flow features but near zero density for the rest of the field. The resulting streamline bundles accentuate flow features while discarding uninteresting regions through *clustered* but not *cluttered* display. Unlike some illustrative techniques that place streamlines very succinctly [19], we allow dense placement of streamlines in critical regions to further strengthen visual perception or impression and give the user the freedom to adjust the density at runtime. Our experience shows that this addition is very effective, especially for viewing 3D flow data sets, as we found out that too few streamlines do not well convey the 3D vector fields perceptually even though the flow structure is captured. We note that if needed, additional seeds can be placed in the flow field to maintain a more uniform distribution of streamlines. Random seeding following a Poisson disk/sphere distribution [37], [43] can be utilized. In some cases, this may help the user gain a better understanding of the overall flow field. In terms of continuity, our seeding algorithm only determines where to drop the seeds and we allow the streamlines to be traced as long as possible. Without enforcing the separating distance between streamlines, the flow patterns revealed in our visualization are thus continuous and complete.

5 Conclusions and Future Work

Effective visualization of three-dimensional flow fields remains a challenge, which is exacerbated by the increase in size and complexity of flow data sets ever produced. The hierarchical streamline bundles we have introduced offer a new way to characterize and visualize the flow structure and patterns in multiscale fashion. Streamline bundles highlight critical points clearly and concisely. Exploring the hierarchy allows a complete visualization of important flow features. Thanks to selective streamline display and flexible LOD refinement, our multiresolution technique is scalable and is promising for viewing large and complex flow fields.

We define flow saliency using the curvature and torsion fields with the goal of highlighting critical points in the saliency map for streamline seeding. The definition of flow saliency can be modified to further categorize the types of critical points or to encompass other flow features of interest, which we would like to explore more. In the future, we will improve flow feature navigation and selection by developing a visual representation similar to the contour tree for abstracting the streamline hierarchy. This addition will provide valuable feedback to the users and guide their exploration: they are able to tell, for example, how many features are left unexplored in a cluster, and decide in advance whether the cluster should be further refined or not. We will apply our technique to other real-world 3D flow data sets and involve domain scientists in the evaluation of this new approach. In particular, we will study parallelization schemes to improve the scalability of our approach and address the large data problem by leveraging the power of parallel heterogeneous systems. Finally, we will extend our approach to 3D vector fields on irregular grids.

Acknowledgments

Our research sponsors include the U.S. Department of Energy, Office of Advanced Scientific Computing Research; the U.S. National Science Foundation through grants IIS-1017935 and OCI-0905008; and Michigan Technological University through a REF Research Seed grant. Sandia National Laboratories is a multiprogram laboratory operated by Sandia Corpration, a Lockheed Martin Company, for the DOE under contract DE-AC04-94-AL85000. Data sets courtesy of the Quake project, the Terascale Supernova Initiative, and the National Center for Atmospheric Research.

References

[1] C. C. Aggarwal and P. S. Yu. Outlier detection for high dimensional data. In *Proceedings of ACM SIGMOD Conference*, pages 37–46, 2001.

[2] K. Bidmon, S. Grottel, F. Bös, J. Pleiss, and T. Ertl. Visual abstractions of solvent pathlines near protein cavities. *Computer Graphics Forum*, 27(3):935–942, 2008.

[3] W. Chen, S. Zhang, S. Correia, and D. S. Ebert. Abstractive representation and exploration of hierarchically clustered diffusion tensor fiber tracts. *Computer Graphics Forum*, 27(3):1071–1078, 2008.

[4] Y. Chen, J. D. Cohen, and J. H. Krolik. Similarity-guided streamline placement with error evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1448–1455, 2007.

[5] I. Corouge, S. Gouttard, and G. Gerig. Towards a shape model of white matter fiber bundles using diffusion tensor MRI. In *Proceedings of International Symposium on Biomedical Imaging*, pages 344–347, 2004.

[6] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, third edition, 1992.

[7] M. Griebel, T. Preußer, M. Rumpf, M. A. Schweitzer, and A. Telea. Flow field clustering via algebraic multigrid. In *Proceedings of IEEE Visualization Conference*, pages 35–42, 2004.

[8] B. Heckel, G. H. Weber, B. Hamann, and K. I. Joy. Construction of vector field hierarchies. In *Proceedings of IEEE Visualization Conference*, pages 19–25, 1999.

[9] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, 2006.

[10] V. Interrante. Illustrating surface shape in volume data via principal direction-driven 3D line integral convolution. In *Proceedings of ACM SIGGRAPH Conference*, pages 109–116, 1997.

[11] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, 1998.

[12] A. K. Jain, M. N. Nurty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[13] R. Jianu, Ç. Demiralp, and D. H. Laidlaw. Exploring 3D DTI fiber tracts with linked 2D representations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1449–1456, 2009.

[14] B. Jobard and W. Lefer. Creating evenly-spaced streamlines of arbitrary density. In *Visualization in Scientific Computing*, pages 43–55, 1997.

[15] G. Kindlmann, R. Whitaker, T. Tasdizen, and T. Möller. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *Proceedings of IEEE Visualization Conference*, pages 163–169, 2003.

[16] R. S. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F. H. Post, and D. Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum*, 23(2):203–222, 2004.

[17] R. S. Laramee, H. Hauser, L. Zhao, and F. H. Post. Topology-based flow visualization, the state of the art. In H. Hauser, H. Hagen, and H. Theisel, editors, *Topology-Based Methods in Visualization*, chapter 1, pages 1–19. Springer Berlin Heidelberg, 2007.

[18] C. H. Lee, A. Varshney, and D. W. Jacobs. Mesh saliency. *ACM Transactions on Graphics*, 24(3):659–666, 2005.

[19] L. Li, H.-H. Hsieh, and H.-W. Shen. Illustrative streamline placement and visualization. In *Proceedings of IEEE VGTC Pacific Visualization Symposium*, pages 79–86, 2008.

[20] L. Li and H.-W. Shen. Image-based streamline generation and rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):630–640, 2007.

[21] Z. Liu, R. J. Moorhead, and J. Groner. An advanced evenly-spaced streamline placement algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):965–972, 2006.

[22] E. B. Lum, A. Stompel, and K.-L. Ma. Using motion to illustrate static 3D shape - kinetic visualization. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):115–126, 2003.

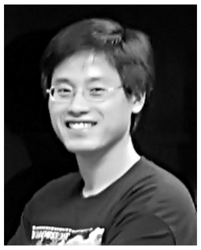
[23] S. Marchesin, C.-K. Chen, C. Ho, and K.-L. Ma. View-dependent streamlines for 3d vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1578–1586, 2010.

[24] T. McLoughlin, R. S. Laramee, R. Peikert, F. H. Post, and M. Chen. Over two decades of integration-based, geometric flow visualization. *Computer Graphics Forum*, 29(6):1807–1829, 2010.

[25] A. Mebarki, P. Alliez, and O. Devillers. Farthest point seeding for efficient placement of streamlines. In *Proceedings of IEEE Visualization Conference*, pages 479–486, 2005.

[26] B. Moberts, A. Vilanova, and J. J. van Wijk. Evaluation of fiber clustering methods for diffusion tensor imaging. In *Proceedings of IEEE Visualization Conference*, pages 65–72, 2005.

[27] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch. The state of the art in flow visualisation: Feature extraction and tracking. *Computer Graphics Forum*, 22(4):775–792, 2003.
[28] W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.
[29] O. Rosanwo, C. Petz, S. Prohaska, H.-C. Hege, and I. Hotz. Dual streamline seeding. In *Proceedings of IEEE VGTC Pacific Visualization Symposium*, pages 9–16, 2009.
[30] T. Salzbrunn, H. Jänicke, T. Wischgoll, and G. Scheuermann. The state of the art in flow visualization: Partition-based techniques. In *Proceedings of Simulation and Visualization Conference*, pages 75–92, 2008.
[31] T. Salzbrunn and G. Scheuermann. Streamline predicates. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1601–1612, 2006.
[32] M. Schlemmer, I. Hotz, B. Hamann, F. Morr, and H. Hagen. Priority streamlines: A context-based visualization of flow fields. In *Proceedings of Eurographics/IEEE VGTC Symposium on Visualization*, pages 227–234, 2007.
[33] B. Spencer, R. S. Laramee, G. Chen, and E. Zhang. Evenly spaced streamlines for surfaces: An image-based approach. *Computer Graphics Forum*, 28(6):1618–1631, 2009.
[34] A. Telea and J. J. van Wijk. Simplified representation of vector fields. In *Proceedings of IEEE Visualization Conference*, pages 35–42, 1999.
[35] H. Theisel and U. Rauschenbach. CurVis - visualizing the curvature of vector fields on the Internet. *Rostocker Informatik-Berichte*, 23:105–114, 1999.
[36] G. Turk and D. Banks. Image-guided streamline placement. In *Proceedings of ACM SIGGRAPH Conference*, pages 453–460, 1996.
[37] V. Verma, D. Kao, and A. Pang. A flow-guided streamline seeding strategy. In *Proceedings of IEEE Visualization Conference*, pages 163–170, 2000.
[38] N. X. Vinh, J. Epps, and J. Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11:2837–2854, 2010.
[39] L. Voinea and A. Telea. Multiscale and multivariate visualizations of software evolution. In *Proceedings of ACM Symposium on Software Visualization*, pages 115–124, 2006.
[40] T. Weinkauf and H. Theisel. Curvature measures of 3D vector fields and their applications. In *Proceedings of International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 121–128, 2002.
[41] D. Weiskopf and G. Erlebacher. Overview of flow visualization. In C. D. Hansen and C. R. Johnson, editors, *The Visualization Handbook*, chapter 12, pages 261–278. Elsevier Academic Press, 2005.
[42] L. Xu, T.-Y. Lee, and H.-W. Shen. An information-theoretic framework for flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1216–1224, 2010.
[43] X. Ye, D. Kao, and A. Pang. Strategy for seeding 3D streamlines. In *Proceedings of IEEE Visualization Conference*, pages 471–478, 2005.
[44] S. Zhang, S. Correia, and D. H. Laidlaw. Identifying white-matter fiber bundles in DTI data using an automated proximity-based fiber-clustering method. *IEEE Transactions on Visualization and Computer Graphics*, 14(5):1044–1053, 2008.



Hongfeng Yu is a postdoctoral researcher at the Combustion Research Facility at Sandia National Laboratories. His research interests include scientific visualization, high-performance computing, and user interfaces and interaction. He received the BE and ME degrees in computer science from Zhejiang University, China, and the PhD degree in computer science from the University of California, Davis. He is a member of the IEEE.

Chaoli Wang is an assistant professor of computer science at Michigan Technological University. His research focuses on large-scale data analysis and visualization, high-performance computing, and user interfaces and interaction. He received the BE and ME degrees in computer science from Fuzhou University, China, in 1998 and 2001, respectively, and the PhD degree in computer and information science from The Ohio State University in 2006. From 2007 to 2009, he was a postdoctoral researcher at the University of California, Davis. He is a member of the IEEE.

Ching-Kuang Shene is a professor of computer science at Michigan Technological University. His research interests include geometric modeling, mesh processing, software visualization, and computer science education. Shene has a PhD degree in computer science from The Johns Hopkins University in 1992, and is a member of ACM, AMS, Eurographics, IEEE/CS, MAA and SIAM.

Jacqueline H. Chen is a Distinguished Member of Technical Staff at the Combustion Research Facility at Sandia National Laboratories. She has contributed broadly to research in petascale direct numerical simulations (DNS) of turbulent combustion focusing on fundamental turbulence-chemistry interactions. These benchmark simulations provide fundamental insight into combustion processes and are used by the combustion modeling community to develop and test turbulent combustion models for engineering CFD simulations. Working closely with SciDAC VACET, SDM, and Ultrascale Visualization Institute, and NCCS/ORNL, she and her team have also developed methodology for automated combustion workflow, in-situ topological feature tracking and visualization of petascale simulated combustion data, and developing DNS software for hybrid architectures. She received the DOE INCITE Award in 2005, 2007, 2008-2010 and 2011 and the Asian American Engineer of the Year Award in 2009. She is a member of the DOE Advanced Scientific Computing Research Advisory Committee (ASCAC) and Subcommittee on Exascale Computing. She was the co-editor of the Proceedings of the Combustion Institute, volumes 29 and 30 and a member of the Executive Committee of the Board of Directors of the Combustion Institute.