A Vocabulary Approach to Partial Streamline Matching and Exploratory Flow Visualization

Jun Tao, *Student Member, IEEE,* Chaoli Wang, *Senior Member, IEEE,*
Ching-Kuang Shene, *Member, IEEE Computer Society,* Raymond A. Shaw

Abstract—Measuring the similarity of integral curves is fundamental to many important flow data analysis and visualization tasks such as feature detection, pattern querying, streamline clustering, and hierarchical exploration. In this paper, we introduce FlowString, a novel vocabulary approach that extracts shape invariant features from streamlines and utilizes a string-based method for exploratory streamline analysis and visualization. Our solution first resamples streamlines by considering their local feature scales. We then classify resampled points along streamlines based on the shape similarity around their local neighborhoods. We encode each streamline into a string of well-selected shape characters, from which we construct meaningful words for querying and retrieval. A unique feature of our approach is that it captures intrinsic streamline similarity that is invariant under translation, rotation and scaling. We design an intuitive interface and user interactions to support flexible querying, allowing exact and approximate searches for partial streamline matching. Users can perform queries at either the character level or the word level, and define their own characters or words conveniently for customized search. We demonstrate the effectiveness of FlowString with several flow field data sets of different sizes and characteristics. We also extend FlowString to handle multiple data sets and perform an empirical expert evaluation to confirm the usefulness of this approach.

Keywords—Streamline similarity, shape invariant features, partial matching, exploratory flow visualization, user interface.

1 Introduction

In flow visualization, measuring the similarity between discrete vectors or the similarity between integral curves is of vital importance for many tasks such as data partitioning, seed placement, field line clustering, and hierarchical exploration. This need has become increasingly necessary and challenging as the size and complexity of flow field data continue to grow dramatically over the years. Early research in this direction focused on vector field similarity and hierarchical classification [7], [19]. This focus has shifted to similarity measurement of integral curves in recent years. Many of the similarity measures designed were targeted on fiber bundle clustering in diffusion tensor imaging (DTI). In this context, spatial proximity is the major criterion for clustering these DTI fiber tracts. Fiber bundles can be formed to characterize different bunches of tracts that share similar trajectories.

In computational fluid dynamics (CFD), integral curves such as streamlines or pathlines traced from flow field data are more complex than DTI fiber tracts. Many CFD simulations produce flow field data featuring regular or turbulent patterns at various locations, orientations and sizes. Clearly, only considering the spatial proximity alone is not able to capture intrinsic similarity among integral curves traced over the field. As a matter of fact, pointwise distance calculation commonly used in proximity-based distance measures is not invariant to translation, rotation and scaling. Although other measures have also been presented that extract features from integral curves and consider feature distribution or transformation for a more robust similarity evaluation, none of them is able to explicitly capture intrinsic similarity that is invariant under translation, rotation and scaling. Furthermore, most of the existing solutions for streamline similarity measurement take each individual streamline of its entirety as the input, measuring partial streamline similarity is not naturally integrated.

In this extended version of our IEEE PacificVis 2014 paper [18], we present FlowString, a novel vocabulary approach for streamline similarity measurement using shape invariant features. Given a flow field data set, we advocate a shape modeling approach to extract different feature characters from the input streamline pool. Each individual streamline is encoded into a string of characters recording its shape features. This draws a clear distinction from existing solutions in that we are now able to perform robust partial streamline matching where both exact and approximate searches are supported. Built on top of the underlying shape analysis framework, we present a user interface that naturally integrates the concepts of characters and words for convenient low-level and high-level streamline feature querying and matching. Our FlowString explicitly categorizes shape features into characters and extracts meaningful words for visual search, which offers a more expressive power for exploratory-based streamline analysis and visualization. The effectiveness of our approach is demonstrated with several flow data sets exhibiting different

- *J. Tao and C.-K. Shene are with the Department of Computer Science, Michigan Technological University, 1400 Townsend Drive, Houghton, MI 49931.*
E-mail: {junt, shene}@mtu.edu.
C. Wang is with the Department of Computer Science & Engineering, University of Notre Dame, Notre Dame, IN 46556.
E-mail: chaoli.wang@nd.edu.
R. A. Shaw is with the Department of Physics, Michigan Technological University, 1400 Townsend Drive, Houghton, MI 49931.
E-mail: rashaw@mtu.edu.

characteristics. Extension of FlowString to handle multiple data sets and feedback from a domain expert are new to the journal version of this work, and thus significantly extends the work in the IEEE PacificVis 2014 paper [18].

2 Related Work

Many similarity measures were presented for clustering DTI fiber bundles and field lines. The spatial proximity between two integral curves is the foundation of many of them [5], [24], [4], [13], [2], [9]. While proximity-based measures are solely based on point locations, other measures extract features from the field or integral curves for similarity analysis. Examples include shape and orientation [3], and local and global geometric properties [16]. Feature distributions of integral curves are less sensitive to noise in the data and sharp turns or twists at certain locations [23], [11], [17], [10], [12]. Therefore, they are often used for more robust similarity measuring. In other approaches, the similarity between integral curves are measured in a transformed feature space [1], [22], [14].

Closely related to our work are the work of Schlemmer et al. [15], Wei et al. [22], Lu et al. [10], and Wang et al. [21]. Schlemmer et al. [15] leveraged moment invariants to detect 2D flow features which are invariant under translation, scaling and rotation. However, their work is restrictive to 2D flow fields and patterns are detected based on local neighborhoods rather than integral curves. Wei et al. [22] extracted features along reparameterized streamlines at equal arc length and used the edit distance to measure streamline similarity. Features of varying scales are only roughly captured by simply recording the length of each resampled streamline. Lu et al. [10] computed statistical distributions of measurements, such as curvature, curl and torsion, along the trajectory to measure streamline similarity. Their approach is invariant to translation and rotation, but not scaling. Wang et al. [21] segmented the streamlines, and applied global alignment to the root segment to match other segments. Their approach could query a pattern including multiple streamline segments. However, the global alignment is needed for every query, which could be costly.

Our FlowString advocates a *shape-based* solution for streamline resampling, feature characterization, and pattern search and recognition. It distinguishes from all previous solutions in that it is specifically designed for robust and flexible partial streamline matching, invariant under translation, rotation and scaling. We enable this through the construction of character-level alphabet and word-level vocabulary. Another distinction is that FlowString is integrated into a user-friendly interface to support intuitive and convenient user interaction and streamline exploration, expressing a more powerful way to visual analytics of flow field data.

3 Terms and Notations

Before giving the overview of our algorithm, we first define the following terms that will be frequently used in the paper:

- *Character:* A character is a unique local shape primitive extracted from streamlines which is invariant to its geometric position and orientation. Characters are the low-level feature descriptors for categorizing streamline shape features.

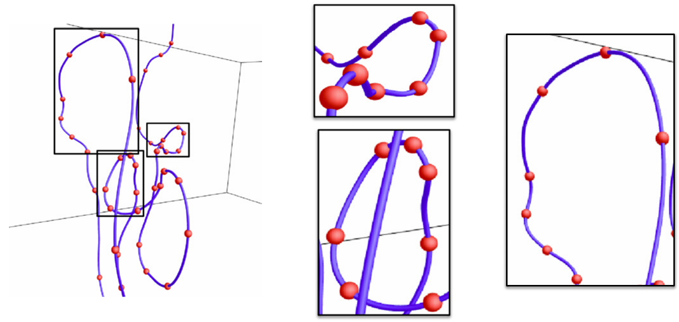


Fig. 1. Resampling a streamline traced from the crayfish data set. The red dots indicate resampled points.

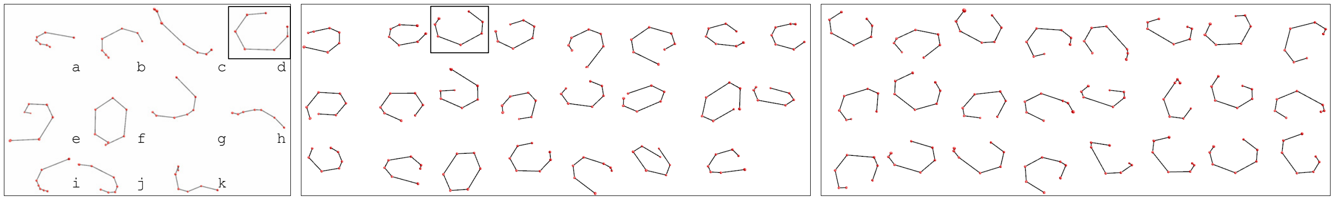
- *Alphabet:* The alphabet consists of a set of characters that describe various local shape features of streamlines traced from a given flow data set.
- *Word:* A word is a sequence of characters encoding a streamline shape pattern. Words are the high-level feature descriptors for differentiating regional streamline shape features.
- *Vocabulary:* The vocabulary consists of a set of words describing various regional shape features of streamlines traced from a given flow data set.
- *String/substring:* A string is the mapping of a global streamline to a sequence of characters. A substring encodes a portion of the corresponding streamline. A substring could match with a word in the vocabulary.

The notations for string operation are mostly consistent with the convention. However, some minor changes are also introduced to adapt to this specific context, which are listed as follows:

- *Character notation:* The shape primitive represented by a character is formed by a set of points in order. A character is denoted as a single lowercase letter `a` (`a'`) if the sample points on the streamline ordered along the flow direction is in the same (reversed) order of the shape primitive. We use the uppercase letter `A` to indicate that the sample points could be in both directions.
- *Multiple characters with common features:* `|` This symbol specifies multiple characters that share some common properties, e.g., ($a_1 | a_2 | \ldots | a_l$) denotes a local shape represented by any character appearing in the parenthesis.
- *Word concatenation:* `|` *and* `&` We use two symbols `|` (`or`) and `&` (`and`) to concatenate two words with the square brackets `[ ]` for distinguishing word boundaries. For example, `[aaa]|[bbb]` returns the segments that match either `aaa` or `bbb`. `[aaa]&[bbb]` finds the segments that contain both `aaa` and `bbb` within some distance threshold.
- *Other symbols*: $^+$, `?`, *and* $\star$ We allow the use of single character repetition $^+$, and wildcard symbols `?` and $\star$. The use of them is consistent with the convention.

4 Our Algorithm

Our FlowString algorithm consists of two major components: *alphabet generation* and *string operation*. Alphabet generation



(a) (b) (c)

Fig. 2. Characters generated from a two-level bottom-up affinity propagation clustering of the crayfish data set. (a) shows the 11 high-level cluster centers, which are assigned to characters `a` to `k` in order. (b) shows the 23 members in the cluster highlighted with a box in (a), which are low-level cluster centers. (c) shows the 24 members in the cluster highlighted with a box in (b).

is to generate the alphabet that describes unique local shape features of streamlines traced from a given flow data set. With this alphabet, we can treat a streamline as a string with a sequence of characters assigned to its sample points. The sample points are resampled from the streamlines based on the local shapes in a preprocessing stage. String operation refers to the matching and querying of the strings based on this alphabet. A suffix tree is built to represent all the strings to enable efficient search and pattern recognition. We automatically extract words from strings to construct the vocabulary to support high-level feature querying.

4.1 Streamline Resampling

For each sample point, its local shape is represented by a set of sample points in its neighborhood with a size of r, i.e., the sample point itself and the $(r-1)/2$ nearest neighbors in both the forward and backward directions along the streamline. The desired streamline resampling should meet two crucial requirements. First, a streamline segment between two sample points should be simple enough, so that no feature is ignored due to under-sampling. Second, since we use a neighborhood of size r to represent the local shape, the density of sample points should be related to the local feature size. That is, for a meaningful comparison, the local features with the same shape should contain the same number of sample points.

Let us consider a continuous 3D curve C and another curve C' which results from uniformly scaling C by a factor s. Let p_1 and p_2 be two points on C, and ${p'}_1$ and ${p'}_2$ be two points on C' which correspond to p_1 and p_2, respectively. The curvature κ' of each point on C' is κ/s, where κ is the curvature of the corresponding point on C. Since the arc length l' between ${p'}_1$ and ${p'}_2$ is $s \times l$, where l is the arc length between p_1 and p_2, the *accumulative curvature* between ${p'}_1$ and ${p'}_2$ is the same as that between p_1 and p_2. This implies that keeping a constant accumulative curvature between two neighboring sample points will produce similar resampling for features with the same shape but of different scales.

For a streamline which is often represented as a polyline, the curvature is not immediately available. Thus, we compute the discrete curvature κ_i at point p_i by

$$\kappa_i = \cos^{-1}(\overrightarrow{p_{i-1}p_i} \cdot \overrightarrow{p_i p_{i+1}}), \tag{1}$$

where p_{i-1}, p_i and p_{i+1} are three consecutive points along the streamline. In other words, the discrete curvature at a point could be approximated by the angle between its two neighboring line segment, and the accumulative curvature becomes the *winding angle* of a streamline segment. Although the winding angle might be affected by the density of points along a polyline, it is very stable if the points traced along the streamline are dense enough.

Our resampling starts from selecting one end of a streamline as the first sample point, and iterates over the other traced points along the streamline. During the iterations, we accumulate the winding angle from the last sample point to the current point. Once the winding angle is larger than a given threshold α, the current point is saved as a new sample point and the winding angle is reset to zero. Note that the neighborhood size r is closely related to the selection of α. That is, when α is smaller, r should be larger to cover the same range of the streamlines in order to capture the shape of local features. If the cumulative winding angle does not reach the threshold for an entire streamline, i.e., mostly a straight line, then we place r sample points evenly on that streamline. In our experiments, we find that setting $\alpha = 1$ (in radian) and $r = 7$ works well for all our test cases. This is because when $\alpha = 1$, the pattern of a streamline segment between two neighboring sample points is relatively simple, and seven consecutive points cover mostly the range of a circle, which is enough to describe a local shape and yet not too complex. Figure 1 shows our resampling result. The three highlighted regions are with three different local scales, which all contain a similar number of sample points after resampling.

4.2 Alphabet Generation

The alphabet generation evaluates the dissimilarity among local shapes of sample points using the *Procrustes distance*, which only considers the shape of objects and ignores their geometric positions and orientations. Then, we apply *affinity propagation* to cluster the local shapes and the cluster centers become the characters.

Dissimilarity Measure. We compute the dissimilarity between the local shapes of two sample points as the Procrustes distance between their neighborhoods, where each neighborhood is a sample point set of size r. Before shape comparison, the two point sets must be superimposed or registered to obtain the optimal translation, rotation, and uniform scaling. This registration is often referred to as the *Procrustes superimposition*. After the superimposition, the two paired

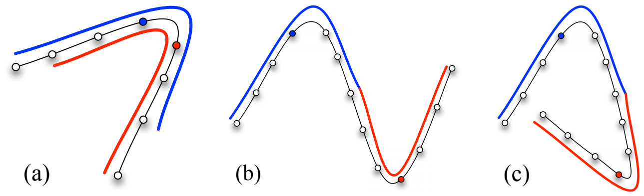


Fig. 3. Character concatenation. The blue and red lines indicate the neighborhoods of blue and red sample points, respectively. (a) characters are assigned to all sample points. $r-1$ sample points are shared by the neighborhoods of blue and red sample points, which produce a deterministic shape. (b) and (c) characters are assigned to every $r-1$ sample points. Only one point is shared by the neighborhoods, which produces different shapes.

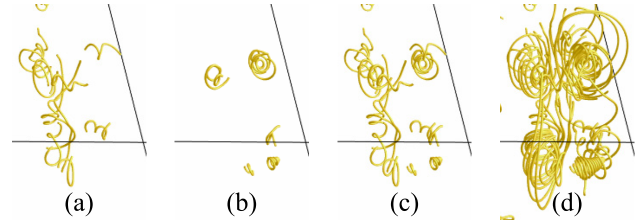


Fig. 4. Matching results using the crayfish data set. A zoomed-in view is used to show a partial volume for clearer observation. (a) and (b) show respectively, exact match results for patterns `EE` and `FF`, where `E` (`F`) is a spiral pattern with large (small) torsion (refer to Figure 2 (a)). (c) and (d) show respectively, exact and approximate ($k = 15$) match results for pattern `(E|F)(E|F)`.

point sets representing the same shape will exactly coincide and thus have the distance of zero. The optimal translation $\mathbf{T}$, rotation $\mathbf{R}$, and uniform scaling s from one point set $P_a = \{p_{a1}, p_{a2}, \ldots, p_{ar}\}$ to another $P_b = \{p_{b1}, p_{b2}, \ldots, p_{br}\}$ are the ones that minimize the summation of the pairwise point distances [8]

$$d = \sum_{i=1}^{r} \left| p_{bi} - p'_{ai} \right|^2, \text{ where } p'_{ai} = s\mathbf{R}p_{ai} + \mathbf{T}. \qquad (2)$$

Note that the minimized d is the Procrustes distance between P_a and P_b. However, in Equation 2, we assume that p_{ai} should be paired with p_{bi}, which might not always be the case for two streamline segments, since two segments with the same shape might be indexed in the opposite directions. Therefore, we compute the distance d_+ between P_a and P_b in the original order and the distance d_- in the reversed order, and use the minimum of d_+ and d_- as the final dissimilarity value.

Affinity Propagation Clustering. Given the dissimilarity measure, we compute the pairwise dissimilarity among all sample points and apply affinity propagation for clustering. The similarity values are then obtained as the negative of the dissimilarity values, as suggested by Frey and Dueck [6]. Unlike k-means and k-medoids clustering algorithms, affinity propagation simultaneously considers all data points as potential exemplars and automatically determines the best number of clusters, with the preference values for each data point as the only parameters. In our scenario, affinity propagation usually generates a fine level of clustering result (with hundreds of clusters). To support pattern query and recognition at a coarser level, the cluster centers at the first level are then clustered by applying affinity propagation again to generate the second-level clusters. In our experiments, the second-level cluster indices serve as the characters, and we find that they already have enough discriminating power.

Figure 2 shows an example of the clustering results. As we can see, the members in the same clusters are usually similar to each other. Although some shapes in the higher level cluster also appear to be similar under this viewing direction, we find that they actually represent different shapes as judged from the query results. For example, they might be portions of spirals with different torsions, which are not revealed under this view.

Character Concatenation. A character corresponding to a sample point determines the local shape of its neighborhood of size r. If a character is assigned to every sample point, a concatenation of two characters represents the shape of a neighborhood of size $r+1$. As shown in Figure 3 (a), this shape is mostly determined by the two characters centering on the blue and red sample points. However, if the characters are only assigned to every $r-1$ points, even if the two characters are exactly the same, the resulting shape of $2r-1$ points could vary significantly. This is because the relative orientation of the two local shapes is undetermined, as shown in Figure 3 (b) and (c). Moreover, since these r points in a neighborhood might not be evenly spaced, the overlapping region of two neighborhoods also decides their relative scale. Also notice that the order of points for each character might affect the shape represented by a string. Certainly, if r is large, we might not need to assign a character to every sample point to maintain the overlapping region of size $r-1$. In practice, since we opt to use a small value for r to avoid too complex local shapes, assigning a character to every sample point seems to be necessary in order to produce deterministic shapes for a string.

4.3 String Operation

Streamline Suffix Tree. After we convert each streamline to a string, we construct a *suffix tree* [20] in linear time and space to enable efficient operations on these strings. A suffix tree is a special kind of tree that presents all the suffixes of the given strings. Each edge of the suffix tree is labeled with a substring in the given strings. For a path starting from the root to any of the leaf nodes, the concatenation of these substrings along this path is a suffix of the given strings.

The problem of search for a string then becomes the search for a node in the suffix tree. Considering that the size of the alphabet is constant, the decision on which edge to visit could be made in constant time, and the search of a string with length m can be performed in $O(m)$ time. Assuming the number of appearance of a string to be searched is z, reporting all the positions of that string takes $O(z)$ time. As a result, with the suffix tree, an exact match of a substring that appears in the given string multiple times only takes $O(m+z)$ time.

Vocabulary Construction. Given a pool of traced streamlines, one interesting yet challenging problem is to automatically identify meaningful words in these streamlines to

TABLE 1
The ten flow data sets and their parameter values. The timing for matrix is the running time for computing pairwise distance among the neighborhoods of sample points. The timing for affinity propagation clustering includes both the first- and second-level clustering. The column "max dist." shows the maximum dissimilarity between any two sample point in that data set.

data set	dimension	# lines	# points	# sample points	# cluster 1st level	# char.	max dist.	timing: matrix	CPU clustering	GPU clustering	setup
vessel	$40 \times 56 \times 68$	100	25606	1338	56	5	31.11	1.45 sec	0.60 min	2.0 sec	0.06 sec
five critical pts.	$51 \times 51 \times 51$	150	18618	1720	75	6	31.06	2.39 sec	1.04 min	3.6 sec	0.07 sec
electron	$64 \times 64 \times 64$	200	24191	1415	38	4	13.98	1.62 sec	0.77 min	3.3 sec	0.05 sec
tornado	$64 \times 64 \times 64$	200	200735	12363	141	6	29.80	126.06 sec	47.07 min	115.0 sec	1.77 sec
two swirls	$64 \times 64 \times 64$	200	209289	13508	156	6	36.05	150.39 sec	86.65 min	247.1 sec	2.07 sec
supernova	$100 \times 100 \times 100$	200	56210	8542	150	7	35.28	35.28 sec	28.85 min	139.0 sec	1.08 sec
crayfish	$322 \times 162 \times 119$	150	164605	7590	178	11	33.77	33.77 sec	21.22 min	89.6 sec	0.97 sec
solar plume	$126 \times 126 \times 512$	200	257087	12484	247	12	36.63	128.47 sec	71.22 min	221.6 sec	2.09 sec
computer room	$417 \times 345 \times 60$	400	361258	9772	262	11	35.91	78.94 sec	36.53 min	75.2 sec	1.43 sec
hurricane	$500 \times 500 \times 100$	200	293572	4766	98	7	35.13	18.42 sec	7.93 min	16.5 sec	0.28 sec

construct the vocabulary. Since the words are depicted by a sequence of characters, we need to not only select representative streamlines, but also extract important segments from them for word identification. With our streamline suffix tree, this could be efficiently solved as we select the most common patterns from the streamlines. In other words, streamline segments that appear most frequently could be identified as words.

We implement our approach on the streamline suffix tree by a simple tree traversal scheme. Since the shape of each streamline segment is captured by a substring in our suffix tree, selecting the common patterns of streamline segments could be considered as the detection of the most frequently appeared substrings. Considering that each potential substring is associated with a node in the suffix tree, the number of appearance for a substring can be efficiently counted with the following two cases:

- If the substring corresponds to a leaf node, its number of appearance is the number of position labels attached to that node.
- If the substring corresponds to an internal node, its number of appearance is the summation of the counts for all the children of that node.

This information could be gathered by a traversal of the tree in the depth-first search manner. Then, all substrings with the length and number of appearance larger than certain thresholds could be reported by another tree traversal. Therefore, identifying words to form the vocabulary can be performed in $O(n)$ time, where n is the total length of the original strings, since the number of nodes is linear to n.

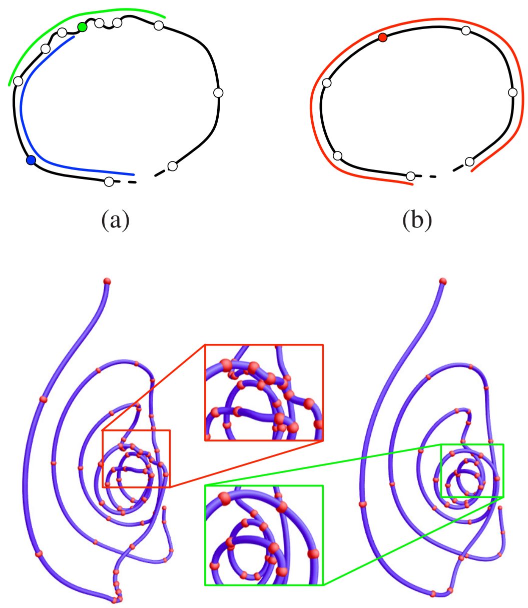
Exact vs. Approximate Search. Since the string is used to represent the shape of streamline segments, exact string matching normally does not provide enough flexibility to capture streamline segments with similar shapes. First, the similarities among the shapes represented by different characters are different, e.g., a portion of spiral with large torsion is more similar to that with small torsion than other shapes. But exact match only produces a binary result, which is either the same or different. Second, with respect to human perception, different numbers of repetition of a certain shape often seem to be similar. For instance, a spiral that contains three circles and another one that contains five circles are usually considered to be similar. Assuming a shape similar to a circle is represented by character `a`, then strings `aaa` and `aaaaa` should be matched in our search. To enable these approximate searches, we implement a dynamic programming approach to detect k-approximate match on the suffix tree, where k is a threshold used in the edit distance. This approach is similar to the traditional computation of edit distance, with the difference that it fills the table when traversing the suffix tree. For implementation detail, we refer readers to our IEEE PacificVis 2014 paper [18].

Figure 4 shows some search results using the crayfish data set, where `E` is a character representing a spiral pattern with large torsion and `F` represents a spiral pattern with small torsion. `E` and `F` correspond to `e` and `f` as shown in Figure 2 (a) respectively. We can observe from Figure 4 (a) that streamline segments matched with `EE` are mostly spirals with large torsion, and those matched with `FF` in (b) are mostly spirals with small torsion. In (c), streamline segments include the results from both (a) and (b). If we enable approximate search, more swirling streamline segments are detected, as shown in (d).

4.4 Further Consideration

High-Level Features. Large-scale shapes or features at higher levels that contain small-scale features could be challenging to detect, due to the extra characters created for small-scale features. For example, in Figure 5 (a), the streamline segment forms a circle, but with a small turbulent portion. Our resampling strategy will densely sample this portion to capture the turbulent feature. This hinders the overall circular shape to be captured. As shown in the Figure 5 (a), neither the neighborhood of the green sample point nor the blue sample point can cover the entire circle. The corresponding shapes to these two sample points are most likely to be identified as a turbulent segment and a hook shape, respectively.

To allow the overall shape of a streamline segment to be correctly understood, we first smooth the streamlines, which removes small-scale features. As shown in Figure 5 (b), the turbulent portion of streamline is smoothed out, so that the circular pattern can be captured at the red sample point. In Figure 5 (c), we demonstrate this with a streamline traced in the crayfish data set. On the left, we can observe that the small



(a) (b)

(c)

Fig. 5. Using smoothed streamlines to capture high-level features. (a) illustration of resampling on original streamlines. (b) illustration of resampling on smoothed streamlines. (c) resampling on a streamline before and after smoothing. The original streamline is shown on the left with a segment highlighted in a red rectangle. The smoothed streamline is shown on the right with the corresponding segment highlighted in a green rectangle.

features create denser sample points on the original streamline. The right one has the streamline smoothed, and the sample points distribute more evenly along the streamline.

In our implementation, we apply a simple Laplacian smoothing for several iterations. In each iteration, we move a point on a streamline towards the center of its two neighbors. More precisely, in each iteration, we update the position of a point p_i with

$$\lambda p'_i + (1-\lambda)\left(\frac{p'_{i-1}+p'_{i+1}}{2}\right), \tag{3}$$

where p'_i, p'_{i-1} and p'_{i+1} are the positions of points p_i, p_{i-1} and p_{i+1}, respectively, and λ is a factor that controls the smoothing speed. The smoothing speed is maximized when $\lambda = 0$, which means that we update the position of p_i with the center of its two neighbors. In this paper, we use a moderate smoothing speed with $\lambda = 0.5$. Once the smoothed streamlines are available, users can choose to include the smoothed streamlines in a query. The query will then be performed on both the original and the smoothed streamlines. Streamline segments matched on the smoothed streamlines will be mapped back to the original streamlines. The matched segments will always be shown in the form of the original streamlines.

Universal Alphabet. The approach described above can be naturally extended to multiple data sets by applying the alphabet generation procedure on multiple streamline sets produced from different data sets. As a result, a universal alphabet will be generated for all the data sets. This is possible because for different data sets or streamline sets, most characters will still be similar, although certain characters might be absent in some data sets due to the lack of corresponding features. In practice, we find that most flow patterns can be captured by a limited set of characters, and the universal alphabet can be generated using a moderate number of data sets that contain various flow features. The universal alphabet is beneficial in two aspects. First, it will eliminate the need to generate an alphabet for each data set, if the basic shape primitives presented in one data set are already captured by the universal alphabet. Second, it will provide a more natural way to compare flow patterns across multiple data sets.

Our universal alphabet is also generated using affinity propagation. Similar to the alphabet generation procedure for a single data set, the universal alphabet is generated in two steps. The first step still computes the first-level cluster centers for each data set independently. Then, we simultaneously consider all the first-level cluster centers as the candidates for the universal alphabet, by computing the dissimilarity values among them and applying affinity propagation for the second-level clustering. Note that we can also generate the universal alphabet in an incremental way using *leveraged affinity propagation*. Assume the data point set is P and the range of possible dissimilarity values is S, the entire dissimilarity matrix can be considered as a mapping $M : P \times P \to S$. Affinity propagation considers all data points at the same time and computes the best exemplars (i.e., clustering centers) from M. Unlike affinity propagation, leveraged affinity propagation samples a subset of data points $P' \in P$ and computes the best exemplars from the mapping $M' : P \times P' \to S$. In each iteration, leveraged affinity propagation keeps the best exemplars from the previous sample points and replaces the other data points with new sample points. This iterative scheme could be applied to extend our alphabet to include extra features from a new data set by a simple modification: in each iteration, we consider the previous universal alphabet and a subset of data points from the new data set as the candidates for the second-level cluster centers, i.e., characters.

The benefit of incremental clustering is mostly on the performance side. However, a data set normally has hundreds of first-level cluster centers. This implies that affinity propagation should be able to handle tens of data sets, which is enough to generate the universal alphabet. Therefore, we prefer affinity propagation that considers all first-level cluster centers at the same time, since it usually yields better clustering results.

4.5 User Interface and Interactions

To make our FlowString a useful tool to support exploratory flow field analysis and visualization, we design a user interface for intuitive and convenient streamline feature querying and matching. Our interface includes four major components: the *alphabet*, *vocabulary*, *query string*, and *streamline* widgets, as shown in Figure 6. These components support visual query and result retrieval.

The alphabet widget visually displays all the characters, as shown in Figure 6 (a). Users can construct a query string from

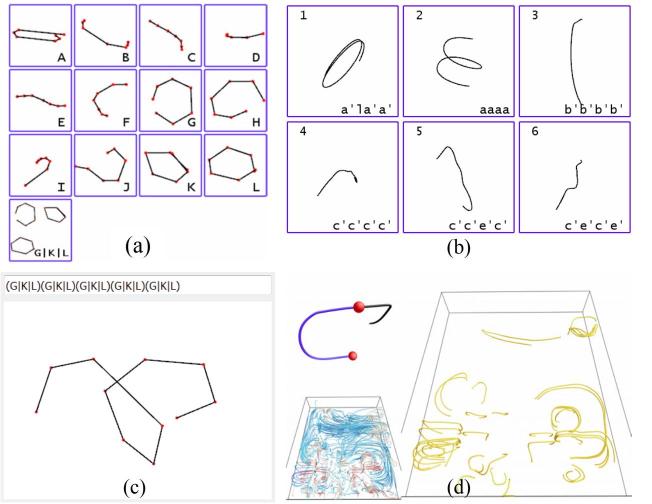


Fig. 6. Alphabet widget (a), vocabulary widget (b), and query string widget (c) with the solar plume data set. Streamline widget (d) with the computer room data set. (a) shows the alphabet visualization where the last character is created by the user to match either `G`, `K` or `L`. (b) shows the first page of the vocabulary widget. (c) shows a query string in the forms of text and polyline. (d) shows the user-selected query segment on the upper-left subwindow (where two red spheres are used to delimitate the blue segment as the query pattern), all streamlines on the lower-left subwindow, and the query result on the right subwindow.

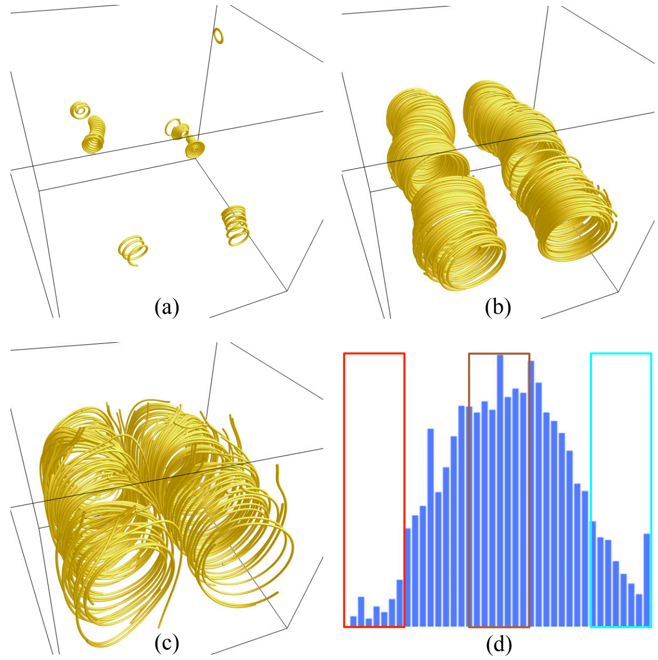


Fig. 7. Circular patterns queried by `MMM` in "universal 1" alphabet (Figure 11 (a)) at the user-specified scale using the two swirls data set. (a) small-scale features. (b) medium-scale features. (c) large-scale features. (d) histogram of feature scales. The red, brown and cyan rectangles represent the selected scale ranges corresponding to (a), (b) and (c), respectively.

this widget by clicking on the displayed characters or typing in an input box. After clicking on each character, the query string and the query result will be updated on the fly. Users can also select multiple existing characters to create a new character, which can match with either of the selected characters. For example, they can select `G`, `K` and `L` to create the character `G|K|L`, as shown in the bottom of (a). Clicking on this new character, the query string widget will append `(G|K|L)` to the current query string. The query string generated from clicking on `(G|K|L)` five times is shown in (c). The vocabulary widget visualizes all the words automatically detected from the streamline suffix tree, as shown in Figure 6 (b). Users can click on a word to retrieve the corresponding pattern in the flow field. They can also select multiple words in sequence to search streamline segments matching with the concatenation of those words. In the first row of Figure 8, we show the selected words in the vocabulary widget and their corresponding query result in the streamline widget. The query string widget displays the query in both textual and visual forms, as shown in Figure 6 (c). Users can freely change the textual query string and its visual form will be updated accordingly. Several sliders are provided to adjust the parameter for k-approximate search, and the thresholds of frequency and length for word generation. In Figure 6 (d), the streamline widget shows the input streamlines at the bottom left, from which users select a streamline. Users can then specify a segment of the streamline to query by moving the two end points, which are shown as the red balls. In this example, a "U"-shape segment is selected, and the query result is shown on the right of this widget.

In addition, we provide a bar chart histogram visualization for users to perceive the scales of matched segments and specify a desired range of scales to further refine the query result. In Figure 7, the circular pattern is queried by `MMM`, and the histogram of scales is plotted in the bar chart as shown in (d). Then, users can brush the histogram to select a range of scales for query. In (a), (b) and (c), we show the matched segments at small, medium and large scales, respectively. The brushed ranges are indicated in (d) by the red, brown and cyan rectangles, respectively. To compute the scale of a matched segment, we follow the optimal scale computation in the registration of two point sets [8]. Formally, considered $P = \{p_1, p_2, \ldots, p_n\}$ to be the sample points on the matched segment, the scale of this segment is given by

$$s = \sqrt{\sum_{i=1}^{n} (p_i - c)^2}, \tag{4}$$

where s is the computed scale and $c = (p_1 + p_2 + \cdots + p_n)/n$ is the center of points in P.

5 Results and Discussion

5.1 Performance and Parameters

Table 1 shows the configurations of ten data sets, the timing for the first- and second-level affinity propagation clustering, and launching the program. For each of the data sets, we randomly placed seeds to trace the pool of streamlines. All the timing results were collected on a PC with an Intel Core

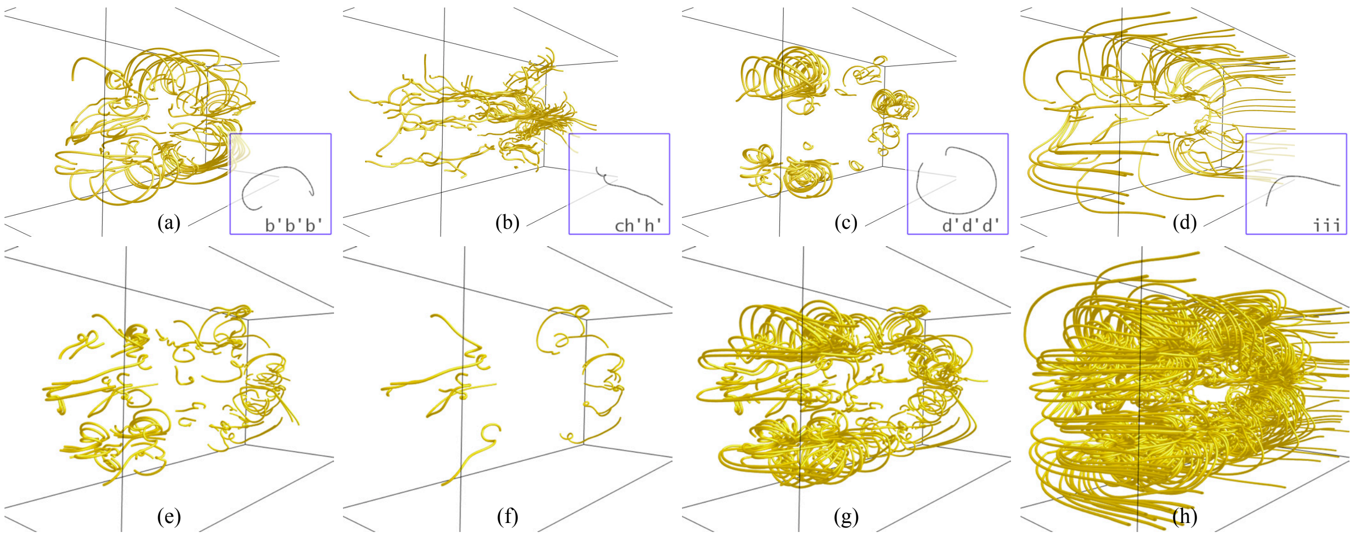


Fig. 8. Case study for the crayfish data set. (a) to (d) show streamline segments matched by four automatically generated words. (e) to (h) show query results of `(A|I)`$^+$`(D|E|F|K)(D|E|F|K)`, `(A|I)(A|I)`$^+$`(D|E|F|K)(D|E|F|K)`, `(A|I)???(D|E|F|K)(D|E|F|K)`, and `(A|I)*(D|E|F|K)(D|E|F|K)`, respectively.

i7-960 CPU running at 3.2GHz, 24GB main memory, and an nVidia Geforce 670 graphics card with 2GB graphics memory.

From the results shown in Table 1, it is obvious that affinity propagation clustering dominates the timing when it is performed on the CPU. We leveraged GPU CUDA to speed up this procedure. For most of the data sets, we performed the clustering by affinity propagation using the GPU. For the solar plume and two swirls data sets, a GPU implementation of *leveraged affinity propagation* was used, since the memory needed to perform affinity propagation exceeds the limit of graphics memory. Unlike affinity propagation which considers all the data points (and the similarities among them) at the same time, leveraged affinity propagation samples from the full set of potential similarities and performs several rounds of sparse affinity propagation, iteratively refining the samples. Thus, the required memory space is reduced with leveraged affinity propagation. The performance of affinity propagation was greatly improved using the GPU. For most of the data sets, the clustering step only took around one minute. For the two swirls data set which contains the most number of sample points, it still could be completed in five minutes. We believe that the timing for clustering is acceptable, since it only needs to run once for a pool of streamlines.

The dissimilarity matrix computation can be performed in reasonable time using the GPU. For the two swirls data set, it took 150 seconds to complete, and the costs for other data sets were even less. Other than these preprocessing steps, the other steps could be finished on the fly. It only took seconds to setup the program for a new run, which includes the time for resampling, computing the dissimilarities between each sample point and each character, and constructing the suffix tree.

Parameter setting is straightforward. The approximation threshold k, minimum number of repetition q, and minimum length and frequency for generating the vocabulary are four parameters that users can configure. They could be adjusted to update the query result in real time. The insertion and deletion costs are automatically decided for each data set. To avoid frequent insertion and deletion, they are both assigned twice the value of maximum dissimilarity between any two sample points in that data set. This rule applies to all the following case studies.

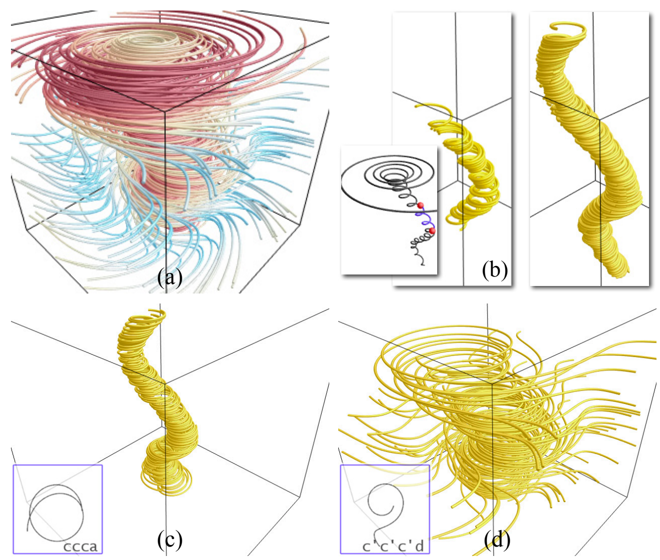


Fig. 9. Case study for the tornado data set. (a) shows all streamlines. (b) shows query results for a user-selected streamline segment with different settings. (c) and (d) show streamline segments matched by two automatically generated words.

5.2 Case Studies

Crayfish. Figure 8 demonstrates query results of both automatically generated words and user inputs using the crayfish data set. In the first row of Figure 8, the four words are selected from a vocabulary of seven words, which are generated with the minimum number of appearance and the minimum

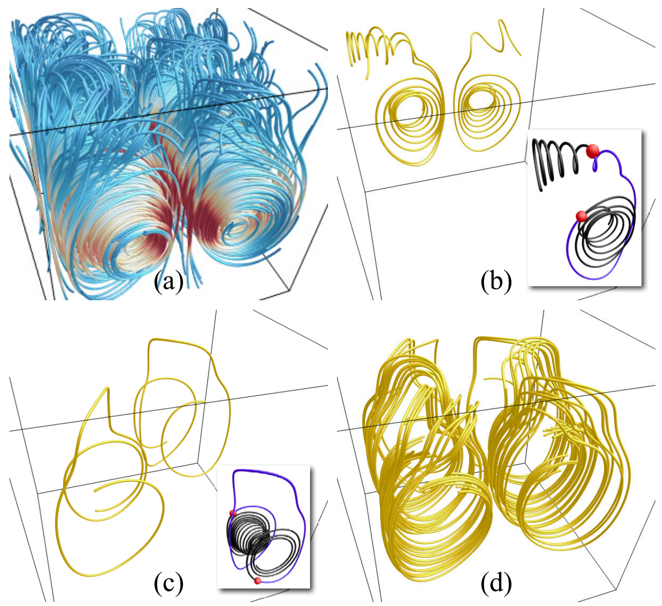


Fig. 10. Case study for the two swirls data set. (a) shows all streamlines. (b) shows the query result for a user-selected streamline segment with the minimum number of repetition $q = 1$. (c) and (d) show query results for a user-selected streamline segment with $q = 0$ and $q = 1$, respectively.

length set to 100 and 3, respectively. We can see that the word `b′b′b′` mostly corresponds to streamline segments of "C"-shape. The word `ch′h′` finds those turbulent segments inside. The word `d′d′d′` matches segments with swirling patterns. Unlike those in Figure 4 (b), `d′d′d′` is usually an elliptical spiral instead of a circular one. Finally, the word `iii` corresponds to streamline segments of "L"-shape on the outer layer along the boundary. We find that most of words with clear patterns are repetitions of a single character. A word with multiple characters often indicates a streamline segment that connects multiple patterns, which is less distinguishable by human observers.

In the second row of Figure 8, we demonstrate an example of using user input to search for a combined pattern that contains a straight segment followed by a spiral pattern. As shown in Figure 2, characters `A` and `I` represent shapes that start with straight segments and `D`, `E`, `F` and `K` are mostly swirling patterns. (e) shows the query result for user input $(\mathtt{A|I})^+$`(D|E|F|K)(D|E|F|K)`, where $^+$ indicates that character `A|I` could repeat multiple times. We then further refine the query result by repeating character `A|I`, which ensures that the straight segment is obvious enough for human perception. As shown in (f), the refined query matches less streamline segments, but the straight segment can be better observed in most of the matched segments. The query `(A|I)???(D|E|F|K)(D|E|F|K)` allows any pattern represented by less than three characters to be inserted between the straight pattern and the swirling pattern, which makes the resulting segments in (g) contain more complex patterns. Finally, if we allow any pattern with arbitrary length to be inserted by querying `(A|I)*(D|E|F|K)(D|E|F|K)`, almost all the input streamlines could be matched, since most of the streamlines contain a straight portion on the outer layer and spirals inside the volume.

Tornado. In Figure 9 (b), the query result on the left is matched by using the exact string `a′bbba′a′bbba′a′a′b`, which corresponds to the user-selected segment. The query result on the right is found by replacing each of the characters with a user-defined character `(A|B|E)`, since these three characters are similar. The exact string matches only the segments that are almost the same as the query segment, while the modified query matches more segments in the core of the tornado. Figure 9 (c) and (d) show the segments corresponding to the words `ccca` and `c′c′c′d`, respectively. Characters `a` and `c` are mostly circles, and character `d` matches the segments with "S"-shape on the outer layer of the tornado. We can observe that when `c` concatenates with `a`, it corresponds to the small-scale circles. When `c` connects with `d`, it matches the large-scale circles. This demonstrates that the scale of a character in a streamline depends on its context, which ensures that the shape for a string is mostly determined.

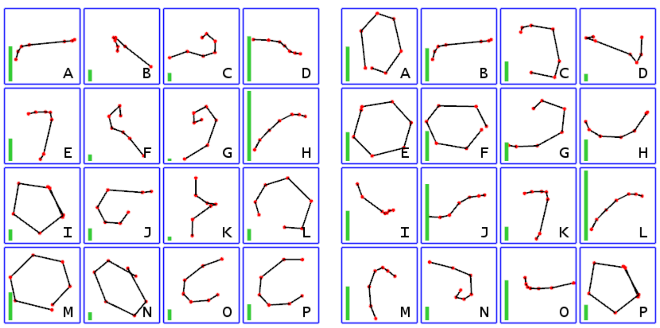
Two Swirls. Figure 10 demonstrates query results of two user-selected streamline segments. In (b), the query segment is one that connects a small spiral pattern and a large swirling pattern. The corresponding query string is `d′d′c′c′c′e′e′a′a′c′d′d′d′`, which matches only the query string itself. The reason is that the query string is somewhat complicated, and even the very similar segments might vary for one or two characters, especially in terms of the number of repetition. We then change the minimum number of repetition q to one, and the query string is modified to $\mathtt{D^+C^+E^+A^+C^+D^+}$. Note that $\mathtt{D^+}$ at the beginning and the end allows the spirals to be displayed in the query result. This query finds two more similar patterns, as shown in (b). In (c) and (d), the query segment is one that connects two large swirling patterns. The query using the exact string `d′c′c′a′aba′a′c′d′d′d′d′d` on that segment finds itself and another very similar one. For the same reason as the previous example, we set $q = 1$. The query string is changed to $\mathtt{D^+C^+A^+B^+A^+C^+D^+}$ accordingly. It matches more segments with the same pattern. In (d), we manually change the query string to $\mathtt{DC^+A^+B^+A^+C^+D}$, which ignores the swirling pattern at the two ends for a clearer observation.

5.3 Universal Alphabet

Qualitative Comparison. Figure 11 demonstrates two examples of universal alphabet when matching with the solar plume data set. Figure 11 (a) shows the universal alphabet generated with all the ten data sets, and Figure 11 (b) shows the universal alphabet generated from five data sets (namely, crayfish, computer room, solar plume, supernova, and two swirl data sets). Data sets used to generate the second universal alphabet are selected according to their coverage of flow patterns. The five selected data sets are more complicated and contain various kinds of flow pattern, so that they are more likely to generate a meaningful universal alphabet. By comparing the shape and frequency of appearance in the solar plume data set for each character, we can observe that the two universal alphabets are actually quite similar. The most

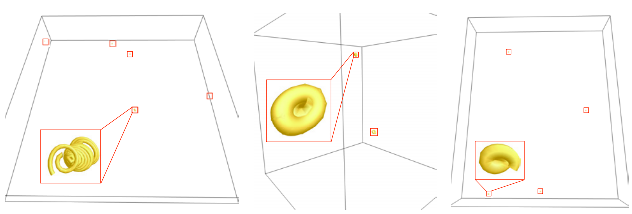
TABLE 2
The ten flow data sets and their average and standard deviation of errors. "universal 1" and "universal 2" correspond to the universal alphabets generated using all the ten data sets and five of the ten data sets, respectively. "single" indicates the alphabet for each data set generated using only that data set.

		vessel	five critical points	electron	tornado	two swirls	supernova	crayfish	solar plume	computer room	hurricane
universal 1	average	7.61	6.90	4.52	5.35	4.48	6.31	7.49	6.35	7.19	5.61
	standard deviation	1.70	2.05	1.62	2.01	2.32	2.36	1.94	1.98	2.57	2.19
universal 2	average	7.59	8.14	5.32	6.65	3.60	6.04	7.50	5.98	7.18	5.35
	standard deviation	1.97	3.28	2.28	2.64	2.53	2.31	2.56	1.98	2.71	2.45
single	average	6.63	6.15	2.56	3.93	3.84	5.1	6.89	5.56	7.61	5.01
	standard deviation	2.54	2.86	1.84	1.94	2.77	2.43	1.92	1.98	2.49	2.54
universal 1 vs. single avg. difference		0.97	0.75	1.96	1.42	0.64	1.25	0.60	0.79	-0.42	0.60
universal 2 vs. single avg. difference		0.96	1.99	2.76	2.71	-0.25	0.99	0.61	0.42	-0.43	0.34



(a) (b)

Fig. 11. Universal alphabets when matching with the solar plume data set. The green bar on the left side of each character indicates its number of appearance in the data set. The alphabets are generated using (a) all the ten data sets and (b) five of the ten data sets, respectively.



(a) (b) (c)

Fig. 12. The small-scale spirals matched by the character `I` in the universal alphabet. The data sets used are (a) hurricane, (b) supernova, and (c) computer room.

common character is `H` in the first alphabet and `L` in the second one. The shapes of these two characters are almost the same. This relationship can be found between `I` and `P`, `M` and `E`, `A` and `B`, `E` and `K`, and `D` and `J`, where the first characters in these pairs are from the first universal alphabet and the second characters are from the second universal alphabet.

In Figure 12, we demonstrate the matching result for `III` using the first universal alphabet with different data sets. The character `I` mostly corresponds to the small-scale spirals. Figure 12 shows that in the hurricane, supernova, and computer room data sets, string `III` finds us the small-scale spirals, which are difficult to notice and locate. However, if the alphabet from a single data set is used, the comparison of the same flow feature across multiple data sets needs to start from the very first step for every data set. Moreover, users will have to make the connection of strings or words across data sets by themselves. Using the universal alphabet, users can simply apply the previous query on the later data sets. Thus, the universal alphabet makes it convenient to compare and explore flow patterns for multiple data sets.

Quantitative Comparison. In addition to qualitative comparison, we also evaluate the effectiveness of different alphabets through quantitative comparison. Table 2 shows the average and standard deviation of errors for each data set using different alphabets. The error is given by the Procrustes distance between the neighborhood of a sample point and the corresponding character. We can observe that the errors using the universal alphabet are usually slightly larger, which is expected. Note that the errors using the alphabet from a single data set is not always smaller due to the fact that affinity propagation is not specifically designed to reduce the within-group variance.

We apply t-tests to evaluate differences between errors using the alphabet "universal 1" and "single", and using the alphabet "universal 2" and "single" for each data set. All the p-values are smaller than 1×10^{-26}, indicating that for each data set significant difference is found between the universal alphabets and the alphabet from a single data set. This is not surprising with the large sample size and relatively small standard deviation. According to the central limit theorem, the estimated variance of sample averages is very small in this case. Thus, even small difference between the sample averages could be significant. However, even if the errors are unlikely to come from the same distribution, the differences between the averages are not large. We can observe in Table 2 that most of the differences are smaller than one, and the largest difference is 2.76 (between "universal 2" and "single" for the electron data set). These value are relatively small compared to the maximum distance between the neighborhoods of any two sample points, which is usually larger than 30 (Table 1). In addition, for the more complicated data sets, e.g., the crayfish, solar plume, computer room, and two swirls data sets, we find that although the errors are usually larger, the differences between errors from universal and single alphabets are actually smaller. For example, the computer room data set has the largest average error with the alphabet from a single data set, but the average errors even decrease with the universal alphabets. In contrast, the data sets that have simple

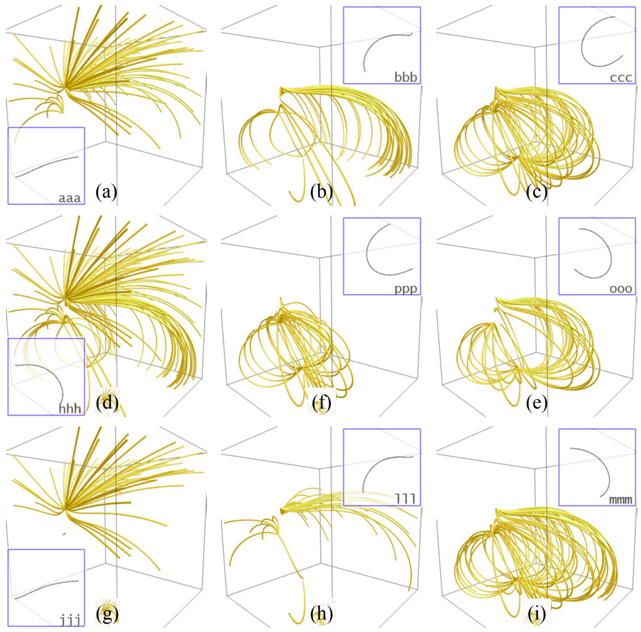


Fig. 13. The words of electron data set using the alphabets generated from itself (first row), from all the ten data sets (second row), and from five of the ten data sets (third row), respectively.

flow patterns, e.g., the electron and tornado data sets, suffer from larger error differences. This is probably due to the fact that the streamlines in these data sets are similar and better captured by a small alphabet generated from a single data set. Thus, they are more likely to be affected when we consider other data sets with different features.

Discriminative Power. Our results show that although the universal alphabet usually produces comparable results for those complicated data sets, the discriminative power of the universal alphabet to simple data sets often reduces. In the first row of Figure 13, we can observe that the alphabet generated from only the electron data set produces a high-quality vocabulary. The word `aaa` finds the mostly straight streamlines; `bbb` matches the dissymmetric curvy streamlines; `ccc` corresponds to the symmetric curvy streamlines with different winding angles. In the second and third rows of Figure 13, six words are shown based on the universal alphabet generated from all the ten data sets and five of the ten data sets, respectively. Although the words still distinguish streamlines with different winding angles to some degree, the discriminative power apparently decreases compared to the alphabet generated from a single data set. This is due to the fact that the streamlines in the electron data set contain only simple patterns. If the electron data set is the only one to generate the alphabet, these patterns could be well captured. However, when other data sets are considered, these clusters have to compromise with other data sets. As a result, the discriminative power for this data set is traded to enhance the overall effectiveness. As we can observe in Figure 13, the features shown in (a) and (b) are merged in (d), when all the data sets are considered. On the other hand, since more complicated data sets already contain various kinds of flow

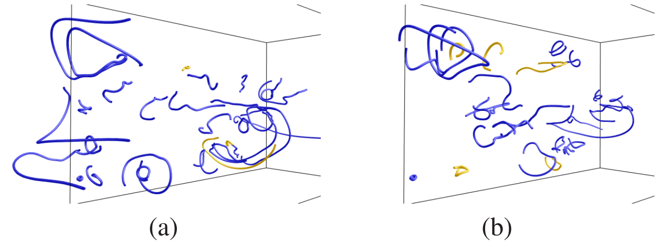


Fig. 14. Patterns matched by `GGG` (a) and `OOO` (b) in "universal 1" alphabet using both the original and smoothed streamlines of the crayfish data set. The yellow segments are matched on the original streamlines. The blue segments are only matched on the smoothed streamlines.

pattern, including other data sets might not introduce new patterns or increase the in-group variance. Therefore, applying our approach on these kinds of data set seems to produce more stable results.

5.4 Smoothed Streamlines

In Figure 14, we query the crayfish data set using "universal 1" alphabet with both the original and smoothed streamlines. To demonstrate the effectiveness, the segments that are only matched on the smoothed streamlines are highlighted in blue. In (a), the queried pattern corresponds to `GGG`, which is a hook shape with a small circle-like pattern at one end. We can observe that the yellow segments matched on the original streamlines have exactly the queried shape, but the blue segments are more likely to contain some turbulent portion of the hook. In (b), a "U"-shape pattern is queried. The segments matched on the smoothed streamlines seem to contain even more diversified shapes. But overall, they are either in the "U"-shape or elongated ellipse shape, which can be considered as the concatenation of two "U"-shape patterns. Actually, this query result depends on the degree that we smooth the streamlines, because this degree determines which features will be smoothed out. If the degree is large, more features will be removed, and the query result will be more diversified when mapped back to the original streamlines.

5.5 Library Release

To reduce the effort for other researchers who would be interested in applying or extending our FlowString approach, we encapsulate our C++ implementation into a library, which is available for download at

`http://www.nd.edu/~cwang11/flowstring.html`.

This library defines a new class called "FlowString" together with various methods. These methods can be applied to encode a pool of streamlines using a universal alphabet and perform queries on the encoded streamlines. All the features of query described in Section 4 are included in this release. The universal alphabet generated from the ten data sets listed in Table 1 is currently available on the same webpage. Further change to this universal alphabet is possible in order to include new flow features. Our ultimate goal is to produce a benchmark field line shape database extracted from various kinds of flow field data sets, i.e., a more complete universal alphabet that could be applied to as many data sets as possible.

5.6 Limitations

Performance. In Table 1, we can see that the affinity propagation clustering dominates the running time, especially using CPU. Utilizing the computation power of GPU, this can be greatly reduced. For any data set listed in Table 1, it takes up to several minutes for the dissimilarity matrix computation and GPU clustering. Users can freely apply any centroid-based clustering on our dissimilarity matrix to achieve better efficiency, thus further improving the performance. On the other hand, with an existing alphabet or using the universal alphabet, our approach is very efficient as the setup time only takes a few seconds.

Meaning of Character. Our approach is based on the similarity measure on the local neighborhood, which does not always conforms to human perception. Characters may be considered to be similar by human if they share certain characteristics. However, the Procrustes distance between the corresponding point sets can still be large. In our current approach, this is solved by allowing users to define a new character for multiple characters with common features. In the future, we may also consider the distance in some feature space, so that the meaning of a character can be better expressed. For example, we may generate a character for all spiral patterns.

Straight Segments. None of the two universal alphabets contains a character representing straight segments, as our resampling strategy will not generate enough sample points on a straight segment. This means that a straight segment is not treated as a standalone feature in our approach. It can only form a character together with the other patterns. For example, a hook shape is a straight segment concatenated with a portion of a circle, as shown by characters `A`, `B` and `E` in "universal 1" alphabet. The term "straight segment" is not well-defined, because it normally depends on the scale of a segment. Specifically, a segment which is considered to be straight usually should be long enough comparing to the pattern that connects to it. Therefore, we believe that combining the straight segment with the other patterns to form characters is reasonable, as this describes the relative size of the straight portion in the character. If there is a specific need to discover the standalone straight segments, a query could be perform to return all segments where the distances between two consecutive sample points are larger than a user-specified value.

6 Empirical Expert Evaluation

To evaluate the effectiveness and learning difficulty of the FlowString, we collaborate with a domain expert in turbulent flow (Dr. Raymond Shaw), whose research focuses on understanding the influence of turbulence on cloud particle growth through condensation and collision. Although three tasks were designed, this study aimed at providing comprehensive reasoning on the effectiveness instead of quantitative results. Dr. Shaw was informed that the comments on reasons behind his rating and selection were more important than the accuracy of tasks. The three tasks are:

- Task 1: In the solar plume data set, find the streamline segments of the small spiral pattern and those of the turbulent flow pattern.
- Task 2: In the crayfish data set, find the streamline segments corresponding to the pattern of a hook connecting with a spiral, and those corresponding to the pattern of small repeated spirals.
- Task 3: In the two swirls data set, find all the common flow patterns.

For the first two tasks, images of the specific flow patterns to find are provided along with text description. For each task, similar questions are asked. These questions can be summarized in three categories:

- Rate the effectiveness in the five-point Likert scale of the vocabulary, approximate search, multiple characters with common features, single character repetition, and wildcard characters.
- Select the most helpful functions to accomplish the tasks.
- Provide detailed comments on the rating and selection.

Comments. After learning the features and interface of the program and practicing on various data sets, Dr. Shaw performed the tasks and provided the following feedback. In general, the FlowString is novel and effective. It provides multiple searching features to identify and locate flow features. The characters successfully capture the basic pattern of flow features. The overlapping of six sample points for two neighboring characters enforces their unique shapes, which is powerful for identifying specific features of interest. In many cases, the repeated use of a single character is very useful for narrowing down the matched results to a specific pattern. In addition, users can define a feature that combines multiple characters which appear similarly. This greatly enhances the ability to locate specific types of flow features. The ability to work with the alphabet, including wildcard characters, allows for great flexibility. Even early in the evaluation the question mark was found to be especially useful in matching a set of characters in a more flexible or general way. For example, if a set of characters was combined so as to search for a specific flow structure of interest, but had become too narrowly defined, inclusion of one or more question marks efficiently allowed the query to become more general. As experience was gained with the range of alphabet capabilities, aspects such as the "prefer user alphabet" proved to be powerful in identifying specific types of features. In particular, the ability to impose directionality on the ordering of the characters is very useful in finding specific geometries of interest.

Visual aspects of the FlowString tool were found to be very effective. Specifically, the interface visualizes characters and words effectively. In this regard, the ability to rotate individual characters in 3D is very important, e.g., for observing the torsion of a spiral. Certain characters, when viewed as 2D projections, initially look like minor variations on a theme, but when viewed in 3D their differences become much clearer. The user interface tools for rotating and viewing shapes are easy to learn. The streamline widget was another graphical interaction tool that proved to be very simple to use and efficient in its ability to allow interaction between the streamlines themselves and the alphabet. In fact it was found that this widget was very useful in "teaching" users how to use the alphabet, e.g., the important aspects like repeated characters.

The vocabulary was found to be one of the most powerful tools, especially for a new user. In effect, the vocabulary has already identified dominant flow structures, even when these features were complex, varying widely in shape and across scales. For example, a search was initiated for what physically could be described as an entrainment event in the crayfish pattern, specifically a long, straight steamline near the outside of the flow, that ends in a tight swirl as it enters the more complex central flow region (e.g., see outer flow features in Figure 8 (h)). Such entrainment events would be typical of a flow pattern of interest in exploring a physical system. Initially the pattern was searched for by using the streamline widget to select a specific example, and then the resulting word was generalized by including wildcard characters, etc. Subsequently, when moving to the vocabulary approach, it was found that a variety of complex but similar streamline patterns were quickly identified, including the same pattern that was originally selected using the streamline widget. Ultimately, the vocabulary proved not only more efficient, but more effective in generalizing the query.

Dr. Shaw further commented that the scale independence of the method is powerful, once fully appreciated. This would be similar to the concept of a wavelet display, in which correlation is shown as a function of position and scale. In terms of learning difficulty, FlowString has sufficient basic features that a user can achieve an impressive range of tasks even after minimal training. FlowString has a range of powerful but more subtle capabilities and benefiting from the full range of these features requires practice and development of experience. Furthermore, it is important to discuss specific features of this tool with an expert for full understanding. The biggest challenge for a scientific user, in his opinion, is the mental picture originally brought to the problem of scale dependence of the flow features and its relationship to streamline sampling resolution. It is crucial to understand that the character matching involves a resampling of seven points, i.e., that the identification of features through cumulative curvature results in the ability to identify similar shapes or features across a wide range of scales. With around an hour of experimenting with FlowString alphabet and vocabulary options the ability to find specific types of features increases rapidly.

Rating and Multiple Selection Questions. Each of the five query features, i.e., the vocabulary, approximate search, multiple characters with common features, single character repetition, and wildcard characters, was rated for each of the tasks. The rating scores of the effectiveness of these features echo these comments. Two features were not rated because they were not used in the tasks. For the thirteen scores in the five-point Likert scale we received, ten of them were rated four points, two were rated five points, and one was rated three points. In the first task, Dr. Shaw felt that every query feature is useful in some aspect, and rated each of them four points. Among these query features, he selected approximate search and single character repetition as the most helpful ones. This might due to the fact that these two features require less experience to use, since the approximate search can reduce the difficulty in composing the exact query string, which is convenient for beginners and the concept of single character repetition is straightforward. In the second task, he rated the approximate search five points and the single character repetition three points. In addition, he selected the alphabet widget to be the most helpful one to compose the query string, and multiple characters with common features and wildcard characters to be the most effective ones to refine the query results. This selection is consistent with the characteristics of the crayfish data set, where each streamline might cross multiple flow features, with less single character repetition. In this case, the multiple characters with common features can group similar flow features, and the wildcard characters can deal with the somewhat turbulent segments connecting the query patterns. This indicates that with only tens of minutes of experience with the tool, users will be able to determine the most effective features to use, even if the use of those features is not trivial. In the third task, Dr. Shaw rated the vocabulary widget to be very useful to find all flow patterns with five points. He did not use the approximate search and wildcard characters in this task, since he was confident to choose which features to use. He selected the vocabulary widget, streamline widget, and multiple characters with common features to be the most helpful features. The streamline widget was used to determine the encoding of a segment when composing the query string. Overall, from these observations, we feel that although users might experience some difficulties in using the tool at the very beginning, they should be able to understand the use of different features and determine the appropriate features according to the given task within one or two hours.

7 Concluding Remarks

We have presented FlowString, a novel vocabulary approach for partial streamline matching for exploratory flow visualization. The unique features of our FlowString are the following: First, our approach supports robust partial matching of streamlines by capturing their intrinsic similarity that is invariant under translation, rotation and scaling. Second, we extract basic shape characters from streamlines to construct an alphabet, from which we detect meaningful shape words to compose a vocabulary to enable both character-level and word-level feature querying and pattern retrieval. Third, we leverage the suffix tree data structure to efficiently speed up both exact and approximate searches, achieving an optimal search efficiency when retrieving multiple occurrences of a single pattern from the streamline pool. Fourth, our method integrates a user-friendly interface for intuitive exploration, allowing users to define their own shape characters and words for customized search. Fifth, our work naturally extends to handle multiple data sets to enable flow feature exploration and comparison across different data sets. To the best of our knowledge, our work is the first one that investigates shape-based streamline similarity measure leveraging the metaphors of characters/alphabet and words/vocabulary. By demonstrating results from different flow field data sets, we show that FlowString represents a new way to flexible streamline matching and expressive flow field exploration.

ACKNOWLEDGMENTS

This research was supported in part by the U.S. National Science Foundation through grants IIS-1017935, DUE-1140512, IIS-1456763, and IIS-1455886.

REFERENCES

[1] A. Brun, H. Knutsson, H.-J. Park, M. E. Shenton, and C.-F. Westin. Clustering fiber traces using normalized cuts. In *Proceedings of International Conference on Medical Image Computing and Computer Assisted Intervention*, pages 368–375, 2004.

[2] 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.

[3] 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.

[4] 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.

[5] Z. Ding, J. C. Gore, and A. W. Anderson. Classification and quantification of neuronal fiber pathways using diffusion tensor MRI. *Magnetic Resonance in Medicine*, 49(4):716–721, 2003.

[6] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315:972–976, 2007.

[7] 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.

[8] B. K. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, 1987.

[9] 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.

[10] K. Lu, A. Chaudhuri, T.-Y. Lee, H.-W. Shen, and P. C. Wong. Exploring vector fields with distribution-based streamline analysis. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 257–264, 2013.

[11] 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.

[12] T. McLoughlin, M. W. Jones, R. S. Laramee, R. Malki, I. Masters, and C. D. Hansen. Similarity measures for enhancing interactive streamline seeding. *IEEE Transactions on Visualization and Computer Graphics*, 19(8):1342–1353, 2013.

[13] 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.

[14] C. Rössl and H. Theisel. Streamline embedding for 3D vector field exploration. *IEEE Transactions on Visualization and Computer Graphics*, 18(3):407–420, 2012.

[15] M. Schlemmer, M. Heringer, F. Morr, I. Hotz, M.-H. 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):1743–1750, 2007.

[16] K. Shi, H. Theisel, H.-C. Hege, and H.-P. Seidel. Path line attributes - an information visualization approach to analyzing the dynamic behavior of 3D time-dependent flow fields. In *Topology-Based Methods in Visualization II*, pages 75–88. 2009.

[17] J. Tao, J. Ma, C. Wang, and C.-K. Shene. A unified approach to streamline selection and viewpoint selection for 3D flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 19(3):393–406, 2013.

[18] J. Tao, C. Wang, and C.-K. Shene. FlowString: Partial streamline matching using shape invariant similarity measure for exploratory flow visualization. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 9–16, 2014.

[19] A. Telea and J. J. van Wijk. Simplified representation of vector fields. In *Proceedings of IEEE Visualization Conference*, pages 35–42, 1999.

[20] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

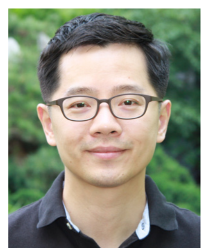
[21] Z. Wang, J. M. Esturo, H.-P. Seidel, and T. Weinkauf. Pattern search in flows based on similarity of stream line segments. In *Proceedings of International Workshop on Vision, Modeling and Visualization*, pages 23–30, 2014.

[22] J. Wei, C. Wang, H. Yu, and K.-L. Ma. A sketch-based interface for classifying and visualizing vector fields. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 129–136, 2010.

[23] 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.

[24] S. Zhang, Ç. Demiralp, and D. H. Laidlaw. Visualizing diffusion tensor MR images using streamtubes and streamsurfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):454–462, 2003.

Jun Tao is a PhD student of computer science at Michigan Technological University. His research interests include flow visualization, image resizing, and mesh editing. He received a BS degree in software engineering from Sun Yat-sen University, China, in 2008, and a MS degree in computer science from Michigan Technological University in 2010. He is a student member of the IEEE.



Chaoli Wang is an associate professor of computer science and engineering at University of Notre Dame. He received a PhD degree in computer and information science from The Ohio State University in 2006. Prior to joining Notre Dame, he was a postdoctoral researcher at University of California, Davis (2007-2009) and an assistant professor of computer science at Michigan Technological University (2009-2014). His main research interest is scientific visualization, in particular on the topics of time-varying multivariate data visualization, flow visualization, and information-theoretic algorithms and graph-based techniques for big data analytics. He is a recipient of the NSF CAREER Award, and has served on the program committees of the IEEE SciVis, EuroVis, and IEEE PacificVis. He is a senior 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.

Raymond A. Shaw is a professor of physics at Michigan Technological University. His research involves the physics of the earth's atmosphere, with an emphasis on clouds and experimental tools for studying clouds. Shaw has a PhD from Pennsylvania State University in 1998, was a postdoctoral fellow in the Advanced Study Program at the National Center for Atmospheric Research, and has been a Humboldt Fellow at the Institute for Tropospheric Research in Leipzig, Germany.