FlowString: Partial Streamline Matching Using Shape Invariant Similarity Measure for Exploratory Flow Visualization

Jun Tao* Chaoli Wang† Ching-Kuang Shene‡

Michigan Technological University

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 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. Leveraging the suffix tree, we enable efficient search of streamline patterns with arbitrary lengths with the complexity linear to the size of the respective pattern. We design an intuitive interface and user interactions to support flexible querying, allowing exact and approximate searches for robust partial streamline similarity 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.

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, 18]. 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 paper, we present FlowString, a novel 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 characteristics.

2 RELATED WORK

Many similarity measures have been presented for clustering DTI fiber bundles and field lines. The spatial proximity between two integral curves is the foundation of many of them [5, 22, 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 [21, 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, 20, 14].

Closely related to our work are the work of Schlemmer et al. [15], Wei et al. [20], and Lu et al. [10]. 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. [20] 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. Our FlowString advocates a *shape-based* solution for streamline resampling, feature characterization, and pattern search and recognition.

*e-mail: junt@mtu.edu

†e-mail: chaoliw@mtu.edu

‡e-mail: shene@mtu.edu

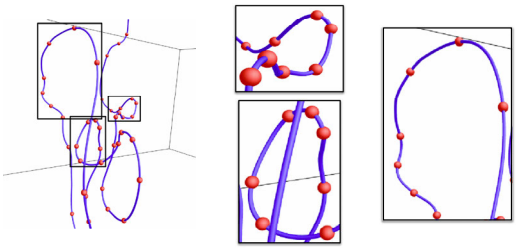


Figure 1: Resampling a streamline traced from the crayfish data set. The red dots are resampled points.

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 nicely integrated into a user 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 describing 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.
- *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 describe 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., $(a_1 \mid a_2 \mid \ldots \mid a_l)$ 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 apart.
- *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 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. 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 Alphabet Generation

Streamline resampling We first resample the streamlines, so that the number of sample points is similar for the local features with the same shape but different scales. 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. Our 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.

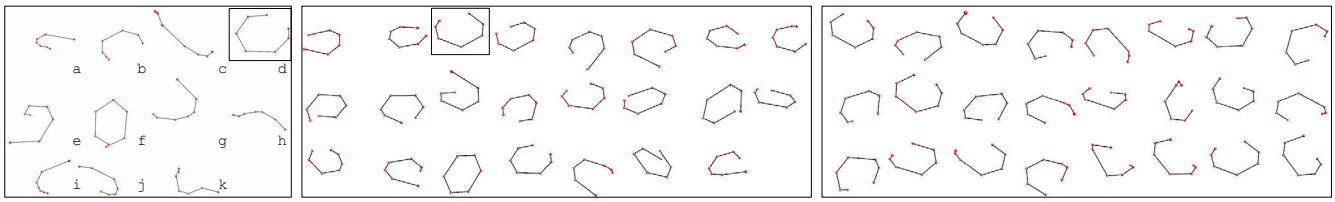
Let us consider a continuous 3D curve C and another curve C' which results from uniformly scaling C by a factor s. Let p_1 and p_2 be two points on C, and p'_1 and p'_2 be two points on C' which correspond to p_1 and p_2, respectively. The curvature κ' of each point on C' is κ/s, where κ is the curvature 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



(a) (b) (c)

Figure 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).

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.

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. This distance only considers the shape of objects and ignores their geometric positions and orientations. Before shape comparison, the two point sets must first 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 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]

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

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, instead of accepting d as their dissimilarity between P_a and P_b immediately, we also compute the distance d' with points being paired in a 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. The algorithm takes these paired similarity values as input, and by simultaneously considering all the data points as the potential cluster centers, it exchanges real-value messages between data points until it converges to a high-quality set of cluster centers. The preference value indicates the probability of selecting the corresponding data point as a cluster center. Using a uniform preference value indicates that all the data points are considered with an equal chance to be cluster centers, and a smaller preference value, i.e., a more negative value in our case, produces a smaller number of clusters. In our scenario, affinity propagation usually generates a fine level of clustering result (with hundreds of clusters). Therefore, we use the minimum of the similarity values as the preference. Although affinity propagation generates high-quality clusters for all the sample points, it is unnecessary to keep the clusters at such a fine level. 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.

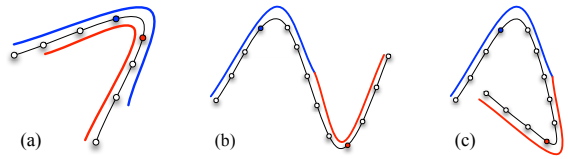


Figure 3: Character concatenation. The blue and red lines indicate the neighborhoods of blue and red sample points, respectively. (a) characters are assigned to all sample points. $r-1$ sample points are shared by the neighborhoods of blue and red sample points, which produce a deterministic shape. (b)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

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. For each sample point and its neighborhood, we select a view that shows its overall shape most clearly by applying a simple heuristic. The camera is set to always look at the center c of this neighborhood, which is calculated by averaging the geometric positions of all points in the neighborhood. The viewing direction is computed as the cross product of two vectors. One of the vectors is $\overrightarrow{v_1} = \overrightarrow{pc}$, which points from the sample point p to the center of the neighborhood c. The other vector is selected as $\overrightarrow{v_2} = \overrightarrow{p_i c}$, where p_i is another sample point in the neighborhood such that $\overrightarrow{v_2}$ is most perpendicular to $\overrightarrow{v_1}$. 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 In our work, a character corresponding to a sample point determines the local shape of its neighborhood of size r. If the characters are 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 characters to every sample point to maintain the

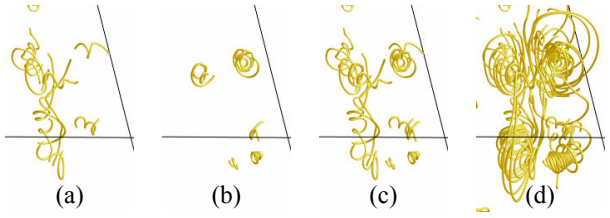


Figure 4: Matching results using the crayfish data set. A zoomed-in view is used to show a partial volume for clearer observation. (a) and (b) show respectively, exact match results for patterns `EE` and `FF`, where `E` (`F`) is a spiral pattern with large (small) torsion (refer to Figure 2 (a)). (c)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

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.

4.2 String Operation

Streamline suffix tree After we convert each streamline to a string, we construct a *suffix tree* [19] 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 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 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 first introduce a straightforward dynamic programming approach to detect k-approximate match on the suffix tree, where k is a threshold used in the edit distance. Then, this approach is extended to support the repetition of a single character.

In our scheme, computing the edit distance of two strings $P = P_1P_2\ldots P_{n_p}$ and $T = T_1T_2\ldots T_{n_t}$ is the same as the traditional approach by filling a table DP of size $n_p \times n_t$, where $\mathrm{DP}[i,j]$ is the edit distance for substrings $P_1\ldots P_i$ and $T_1\ldots T_j$. Our straightforward k-approximate match on a suffix tree traverses the tree in the depth-first search manner and expands the table column by column using the following rule

$$\mathrm{DP}[i,j] = \min\begin{pmatrix} \mathrm{DP}[i-1,j]+\mathrm{cost}_d, \\ \mathrm{DP}[i,j-1]+\mathrm{cost}_i, \\ \mathrm{DP}[i-1,j-1]+\mathrm{cost}_r(P_i,T_j) \end{pmatrix}, \quad (3)$$

where P is the query string to match, T is the string labeled on the path from the root to the currently visited edge or node, cost_d and cost_i are deletion and insertion costs which are set to the maximum dissimilarity value among characters, and $\mathrm{cost}_r(P_i,T_j)$ is the replacing cost which is set to the dissimilarity between characters P_i and T_j. Note that the replacing cost between the same characters is zero, since they represent the same shape. During the traversal, each time we expand the string T by one character along the edge being visited and fill the column $\mathrm{DP}[\ldots,n_t]$. If $\mathrm{DP}[n_p,n_t]$ is smaller than k, then we find a match T whose edit distance to P is within k. Note that we only need to traverse to a certain depth whose corresponding label string is shorter than $n_p + k/\mathrm{cost}_i$, since otherwise we would have an edit distance larger than k. The benefit of implementing approximate search on the suffix tree is that we only need to compute the edit distance once between P and all the appearances of the same substring. Furthermore, if the traversal finishes exploring a branch under a node u and starts to traverse another branch, the columns in the table DP representing the edit distance between P and the label string on the path from the root to u could also be reused.

The special symbols for single character repetition $^+$ and multiple characters with common features `|` are implemented by extending the traditional edit distance. For single character repetition, a minimum number of repetition q is used to guarantee that the pattern is significant enough for human perception. For example, if $q = 3$, `aaa` is considered to be the same as `aaaaa` but not `aa`. This is implemented by replacing any repetition of `a` with length larger than three by `aaa`$^+$. By considering `a`$^+$ as another character, the insertion cost of $T_j =$ `a` when matching with character `a`$^+$ should be zero. This could be formulated as

$$\mathrm{cost}_i(P_i,T_j) = \begin{cases} 0, & \text{if } P_i = T_j{}^+ \\ c, & \text{otherwise} \end{cases}, \quad (4)$$

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
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

Table 1: The eight 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 levels clustering. The column "max dist." shows the maximum dissimilarity between any two sample point in that data set.

where c is some constant. We use Equation 4 to replace the insertion cost in Equation 3, and terminate a searching branch only if all elements in a column DP$[\,, j]$ are larger than k.

For multiple characters with common features, we expand the replacing cost to achieve this goal. That is, we create a new character A with its replacing cost to other characters defined as

$$\text{cost}_r(P_i = \mathtt{A}, T_j) = \begin{cases} 0, & \text{if } T_j \in (\mathtt{a}_1 \mid \mathtt{a}_2 \mid \ldots \mid \mathtt{a}_l) \\ \min \begin{pmatrix} \text{cost}_r(\mathtt{a}_1, T_j), \\ \ldots, \\ \text{cost}_r(\mathtt{a}_l, T_j) \end{pmatrix}, & \text{otherwise} \end{cases} . \tag{5}$$

We use Equation 5 to replace the replacing cost in Equation 3. 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).

4.3 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 consists of four widgets that visualize respectively, the *alphabet*, *vocabulary*, *query string*, and *streamline* widgets. The alphabet widget visually displays all the characters, as shown in Figure 5 (a). Users can construct a query string from this widget by clicking on the displayed characters or typing in an input box. The query result will be updated on the fly. They can also select multiple existing characters to create a new character, which can match with either of the selected characters. The vocabulary widget visualizes all the words automatically detected from the streamline suffix tree, as shown in Figure 5 (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. The query string widget displays the query in both textual and visual forms, as shown in Figure 5 (c). Several sliders are provided to adjust the parameter for k-approximate search, and the thresholds of frequency and length for word generation. The streamline widget shows all the input streamlines and the query result as streamline segments, as shown in Figure 5 (d). It allows users to select a streamline segment from the streamline view and to search for similar segments.

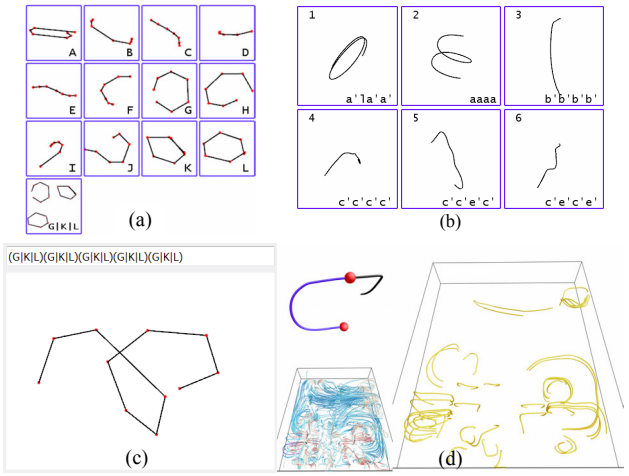


Figure 5: Alphabet widget (a), vocabulary widget (b), and query string widget (c) with the solar plume data set. Streamline widget (d) with the computer room data set. (a) shows the alphabet visualization where the last character is created by the user to match either G, K or L. (b) shows the first page 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.

5 Results and Discussion

5.1 Performance and Parameters

Table 1 shows the configurations of eight 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 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 min-

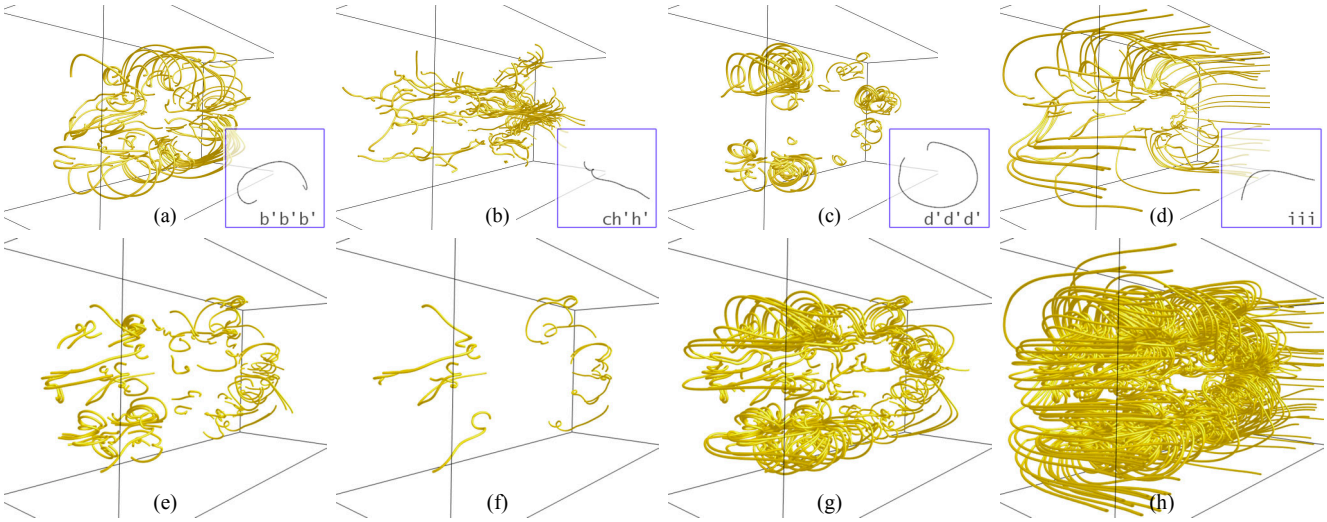


Figure 6: Case study for the crayfish data set. (a) to (d) show streamline segments matched by four automatically generated words. (e) to (h) show query results of `(A|I)+(D|E|F|K)(D|E|F|K)`, `(A|I)(A|I)+(D|E|F|K)(D|E|F|K)`, `(A|I)???(D|E|F|K)(D|E|F|K)`, and `(A|I)*(D|E|F|K)(D|E|F|K)`, respectively.

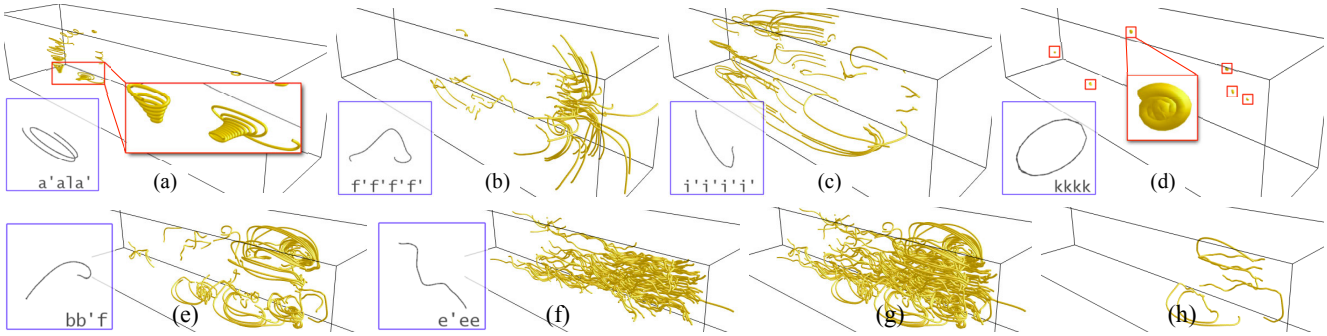


Figure 7: Case study for the plume data set. (a) to (d) show streamline segments matched by four automatically generated words. (e) to (h) show query results when the previously generated alphabet is used for newly traced streamlines. (e) and (f) show streamline segments matched by two words. (g) and (h) show the concatenation of the two words using `|` and `&`, respectively.

utes. We believe that the timing for clustering is acceptable, since it only needs to run once given 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 easily 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.

5.2 Case Studies

Crayfish Figure 6 demonstrates query results of both automatically generated words and user inputs using the crayfish data set. In the first row of Figure 6, the four words are selected from a vocabulary of seven words, which are generated with the minimum number of appearance and 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 6, 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 line segments and `D`, `E`, `F` and `K` are mostly swirling patterns. (e) shows the query result for user input `(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

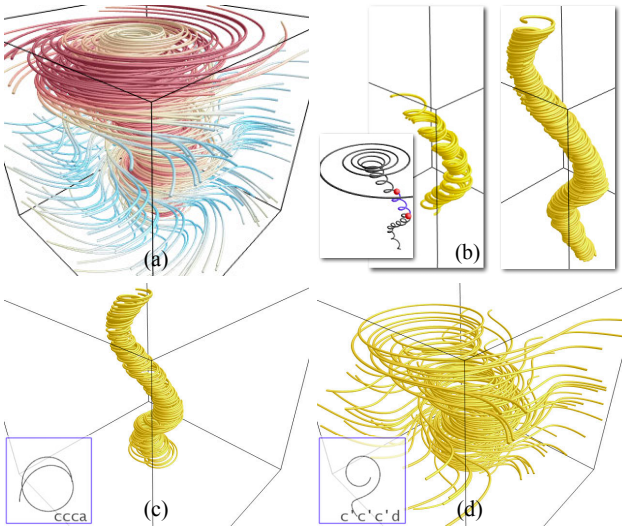


Figure 8: Case study for the tornado data set. (a) shows all streamlines. (b) shows query results for a user-selected streamline segment with different settings. (c) and (d) show streamline segments matched by two automatically generated words.

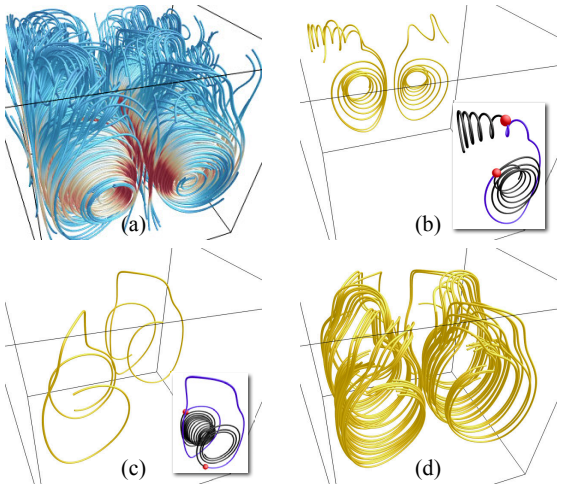


Figure 9: Case study for the two swirls data set. (a) shows all streamlines. (b) shows the query result for a user-selected streamline segment with the minimum number of repetition $q = 1$. (c) and (d) show query results for a user-selected streamline segment with $q = 0$ and $q = 1$, respectively.

`(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.

Solar plume Figure 7 shows query results using the solar plume data set. In the first row of Figure 7, we select four words from a vocabulary of 28 words, which are generated with the minimum number of appearance and length set to 35 and 4, respectively. We can observe that the word `a'ala'` matches very elongated ellipses. The word `f'f'f'f'` finds those segments containing a "L"-turn which mostly reside in the head of solar plume. The word `i'i'i'i'` corresponds to the hook-like segments surrounding the outer layer. We also find six small-scale spirals using the word `kkkk`.

The second row of Figure 7 shows a query example using the previously generated alphabet for a new set of streamlines. We compute the dissimilarity value between each character and each sample point by measuring their Procrustes distance. The character with the minimum dissimilarity to a sample point is assigned to that point. The vocabulary is extracted from the strings corresponding to the new set of streamlines. In (e) and (f), the streamline segments are found by two words `bb'f` and `e'ee`, respectively. The word `bb'f` corresponds to segments with elongated "C"-shape and `e'ee` matches the turbulent segments in the core region. (g) demonstrates the query result for `[bb'f]|[e'ee]`, which contains all the segments matched by both `bb'f` and `e'ee`. We can also search for a pattern of a "C"-shaped segment followed by a turbulent one by applying the query `[bb'f]&[e'ee]`, as shown in (h). There are not many matched segments, since the length of any substring between the two words is constrained.

Tornado In Figure 8 (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 by 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 8 (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 9 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 $\mathtt{D}^+\mathtt{C}^+\mathtt{E}^+\mathtt{A}^+\mathtt{C}^+\mathtt{D}^+$. 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 $\mathtt{D}^+\mathtt{C}^+\mathtt{A}^+\mathtt{B}^+\mathtt{A}^+\mathtt{C}^+\mathtt{D}^+$ accordingly. It matches more segments with the same pattern. In (d), we manually change the query string to $\mathtt{DC}^+\mathtt{A}^+\mathtt{B}^+\mathtt{A}^+\mathtt{C}^+\mathtt{D}$, which ignores the swirling pattern at the two ends for a clearer observation.

6 Future Work

We would like to improve our FlowString in the following ways: First, our current solution that generates an alphabet for a set of streamlines could be extended to create a universal alphabet to handle many data sets. As shown in Figure 7 (e) to (h), once the alphabet is generated, it could be applied to different sets of streamlines traced from the same data set. Moreover, for different data sets, most characters are still similar, except that some characters might

be absent due to the lack of corresponding features in a data set. These facts imply that creating a universal alphabet across multiple data sets is possible.

Second, we feel that string patterns and their meanings could be better analyzed. Although our current implementation only deals with the repetition of a single character, the repetition of a substring should be handled similarly. For example, two streamline segments corresponding to `abcabcabc` and `abcabcabcabc` might be very similar perceptually. A third streamline segment labeled with `abcadcadcabc` could also be similar to the previous two, if the shape represented by `d` is similar to that of `b`. Thus, the detection of approximate substring repetition is important for understanding the pattern of a streamline. The detection of tandem repeats has been studied extensively in the field of computational biology, which could probably suggest a solution to our problem. More sophisticated approaches in string pattern research should be incorporated to better perceive the meaning of strings and further understand the pattern of streamlines.

Third, our current approach should also be improved to query across multiple resolutions. Although our resampling enables scale-invariant query, it might over emphasize small-scale features when users exam patterns at a larger scale, since the small-scale features could contain the same number of characters. For instance, if a circle is represented by `aaaaaaa`, then a circle with a small bump on it might be labeled with `aabbbaaaaa`, where `bbb` corresponds to the small bump. In this case, the small bump introduces considerable change to the string, but does not change the overall shape of the circle, which can hardly be distinguished when we observe it at a coarser level. Extending the string to encode streamlines in multiple resolutions is beneficial, so that such small bumps could be ignored when we study the streamlines at a larger scale.

7 Conclusions

We have presented FlowString, a novel solution 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 nicely integrates a friendly user interface for intuitive exploration, allowing users to define their own shape characters and words for customized search. 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.

Acknowledgements

This work was supported in part by the U.S. National Science Foundation through grants IIS-1017935, DUE-1105047, CNS-1229297, and IIS-1319363. We would like to thank the anonymous reviewers for their insightful comments.

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*.

[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 *Proceedings of Topology-Based Methods in Visualization*, 2007.

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

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

[20] 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.

[21] 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.

[22] 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.