Extracting Flow Features via Supervised Streamline Segmentation

Yifei Li[a], Chaoli Wang[b], Ching-Kuang Shene[a]

[a]*Michigan Technological University*
[b]*University of Notre Dame*

Abstract

Effective flow feature extraction enables users to explore complex flow fields by reducing visual clutter. Existing methods usually use streamline segmentation as a preprocessing step for feature extraction. In our work, features are directly extracted as a result of streamline segmentation. In order to achieve this, we first ask users to specify desired features by manually segmenting a few streamlines from a flow field. Users only need to pick the segmentation points (i.e., positive examples) along a streamline, remaining points will be used as negative examples. Next we compute multiscale features for each positive/negative example and feed them into a binary support vector machine (SVM) trainer. The trained classifier is then used to segment all the streamlines in a flow field. Finally, the segments are clustered based on their shape similarities. Our experiment shows that very good segmentation results can be obtained with only a small number of streamlines to be segmented by users for each data set. We also propose a novel heuristic based on the minimum bounding ellipsoid volume to help determine where to segment a streamline.

Keywords: Flow visualization, Flow feature extraction, Streamline segmentation, Support vector machine

1. Introduction

Flow visualization has been a central topic in scientific visualization for more than two decades. A flow field can be visualized using different techniques, including glyph-based [1], texture-based [2], integration-based [3], partition-based [4], illustration-based [5], and surface-based [6] approaches. Among these techniques, integration-based flow visualization is most widely used in practice. For integration-based flow visualization, particles or seeds are placed in a vector field and advected over time. The traces or field lines that the particles follow, i.e., streamlines for steady flow and pathlines for unsteady flow, depict the underlying vector data. In this paper, integration-based technique with random seeding is used for visualization.

However, visual clutter and occlusion is a major issue when hundreds or thousands of streamlines are rendered to depict a flow field. This makes it difficult for users to explore the interesting features. Although clustering and displaying the streamlines based on their similarities may alleviate this issue, a more subtle issue is that very often not all parts of a streamline are equally important: the part of a streamline in the vicinity of a vortex is more important than the part running through a region of laminar flow. Furthermore, different domain experts may have their own criteria on what constitutes an "interesting flow feature". This observation inspired us to segment a streamline based on *user-defined* features. To the best of our knowledge, this problem has not been well studied by the flow visualization community.

To address this problem, we propose a supervised streamline segmentation algorithm which allows the extraction of user-defined flow features. For each data set, users are required to manually segment only a small number of streamlines in
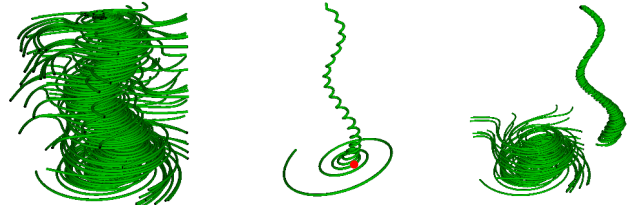

Figure 1: Given an input pool of streamlines (left), we first segment each streamline using our previously learned classifier for segmentation points (middle, the red point is the segmentation point found by our algorithm). Partial streamline features specified by users will be clustered based on their similarities (right).

order to define what flow features they want to extract from the flow field. The user-picked segmentation points along a streamline will be used to generate *positive* training examples, whereas the remaining ones are used to generate *negative* training examples. Multiscale feature vectors are computed for each positive and negative example, and fed into a binary support vector machine (SVM) trainer. Finally, we use the trained classifier to determine the segmentation points for all the streamlines in the data set. A post-processing step is required for grouping nearby segmentation points detected by the classifier. Figure 1 shows an example of extracting user-defined partial flow features.

The remainder of this paper is structured as follows. Section 2 reviews approaches that are most related to our work. Section 3 presents the details of our algorithm and is divided into the following subsections: Section 3.1 introduces basic concepts of supervised learning and SVM, Section 3.2 discusses the features we use to train a classifier, Section 3.3 explains how the positive and negative examples are generated,

Section 3.4 explains how the training is conducted, and Section 3.5 introduces our segmentation algorithm and necessary post-processing steps. Section 4 demonstrates the utility of our algorithm by clustering streamline segments using different 3D flow fields. Section 5 compares our method with a few other state-of-the-art streamline segmentation/feature extraction methods. Finally, Section 6 points out the directions of our future work.

2. Related Work

Flow feature extraction provides an effective way to reduce visual clutter. For 2D flows, Schlemmer et al. [7] and Bujack et al. [8] both leveraged moment invariants to detect 2D flow features. Wei et al. [9] relied on user-sketched 2D curves to retrieve similar occurrences from a 3D flow field. The retrieval might be ambiguous because the 3D streamlines first need to be projected to 2D curves for similarity comparison. Tao et al. [10] converted each streamline into a string such that all the streamlines can be recorded in a suffix tree. The string patterns detected in the suffix tree correspond to certain flow features. Users can specify a query string to search for interesting flow features. Finally, Wang et al. [11] proposed an example-based flow pattern search approach for the detection of similar flow feature patterns given a query pattern, where flow patterns are given by a subset of segments from the set of all streamline segments.

Streamline clustering and selection provides another way to reduce visual clutter. Common clustering algorithms such as nearest neighbor, fuzzy clustering and hierarchical clustering have been used in the works of [12, 13, 14]. Different streamline similarity measures have been proposed for streamline clustering. Examples include the average of point-by-point distance [13], the mean of closest point distances [15] and the thresholded average distance [12]. These similarity measures are all based on the Euclidean distance, and hence are not affine invariant. To overcome this shortcoming, similarity measures based on feature distribution were adopted in the works of [16, 17, 18]. For a detailed survey of streamline clustering methods and similarity metrics, we refer readers to [19]. Günther et al. [20] rendered streamlines using different opacity values which are computed to optimize the balance between information presentation and occlusion avoidance. They also gave a comparison among the different state-of-the-art streamline selection algorithms.

Streamline segmentation has been used to facilitate streamline similarity comparison [17] and flow pattern extraction [10, 11]. Lu et al. [17] recursively segmented a streamline into two most dissimilar segments until either the dissimilarity is below a certain threshold or the current segment is too short. Tao et al. [10] segmented a streamline such that the accumulated curvature of each segment does not exceed a certain threshold. An obvious drawback of their approach is that they failed to separate straight segments. Wang et al. [11] partitioned a streamline into so-called minimal segments first, and the final segmentation is obtained after merging the minimal segments based on two thresholds: total curvature and average binormal
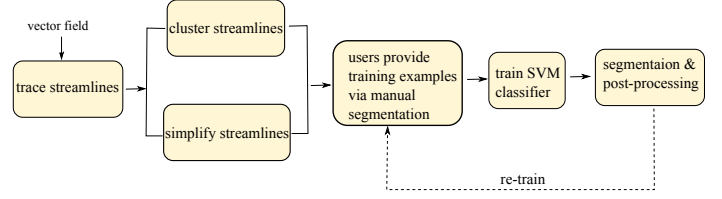

Figure 2: Our proposed supervised streamline segmentation framework

direction. However, their approach cannot segment turbulent streamlines very well. All of these segmentation algorithms need some manually-tuned thresholds to determine whether to segment or not at a given point.

The problem of curve segmentation has also been studied in the computer vision community. However, they usually focused on segmenting a curve into a combination of representations such as lines, circular, elliptical and superelliptical arcs, and polynomials [21]. We cannot apply their methods to streamline segmentation because we are usually interested in more complicated features such as spirals rather than just, for example, circular curves.

The *minima rule* [22] from cognitive science is widely used by mesh segmentation algorithms [23]. The rule states that human beings tend to divide a *surface* into parts at loci of negative minima of each principal curvature along its associated family of lines of curvature. However, after extensive research, we have not found any rules from cognitive science which can help us segment a 3D curve (e.g., streamlines).

3. Supervised streamline segmentation

In order to obtain a streamline segmentation based on user-defined features, we leverage supervised learning to train a classifier to determine whether we should separate a streamline around a given point. The motivation for using machine learning is that we want to take multiple features into consideration when determining whether to segment and generate a complex decision function based on those features.

Therefore, we propose a user-guided streamline segmentation framework, which is illustrated in Figure 2. After streamlines are traced, we cluster (Section 3.3.1) and simplify (Section 3.3.2) all the streamlines. From cluster representatives, users choose which streamlines to segment manually. A binary SVM classifier (Section 3.1) is trained (Section 3.4) to classify segmentation points of a streamline. A post-processing step (Section 3.5) is carried out to generate final segmentation. The training process (the dashed line in Figure 2) can be repeated if users are not satisfied with the segmentation results.

3.1. Support vector machine

The goal of a *supervised learning* problem is to learn a function $y(\mathbf{x})$ given a set of *training examples* $\{(\mathbf{x_1}, t_1), \cdots, (\mathbf{x_N}, t_N)\}$, where $\mathbf{x_i}$ is a *feature vector* and t_i is a *target value*. It is up to applications to determine what a feature vector consists of. The value of t_i is usually $+1$ or -1 for *binary classification* problems. For a binary classification problem, a
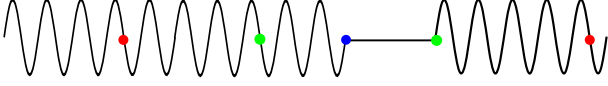
Figure 3: Importance of neighborhood size: the blue point may be considered as a segmentation point if we compare its two neighboring segments whose end points are marked as green because the segments have different "complexity". However, this difference becomes smaller if a larger neighborhood size is considered (marked by red points).

training example $(\mathbf{x_i}, t_i)$ is called a *positive* training example if $t_i = 1$, and a *negative* training example if $t_i = -1$. The precise form of the function $y(\mathbf{x})$ is determined during the *training* phase. The ability to categorize correctly new examples that differ from training examples is known as *generalization*. For more details on supervised learning, please refer to [24].

For *linearly separable* training examples, support vector machine (SVM) [25] learns a linear model of the form $y(\mathbf{x}) = \mathbf{w}^T \cdot \mathbf{x} + b$ for binary classification problems. The corresponding decision function is $f(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$, where $\mathbf{w}$ and b are the normal and the intercept of a hyperplane. In other words, it tries to find a hyperplane to separate positive examples from negative ones. Among all the hyperplanes separating the data, there exists a unique *optimal hyperplane*, distinguished by the maximum *margin* of separation between any training point and the hyperplane. The margin is the distance of the closest point to the hyperplane. The points closest to the hyperplane are called *support vectors*.

For non-linearly separable training examples, a *kernel trick* is applied which transforms the training data $\mathbf{x}$ into a high-dimensional feature space $\phi(\mathbf{x})$ such that a separating hyperplane can be found. A commonly-used kernel function is the radial basis function (RBF) $\exp(-\|\mathbf{x} - \mathbf{x}'\|^2/2\sigma^2)$, where σ is a free parameter.

3.2. Features vectors

Intuitively, the streamline segments on two sides of a segmentation point should "look" differently, hence we focus on the metrics which evaluate the similarity between two neighboring segments. We also need to determine the size of neighborhood to be considered during similarity comparison. The importance of neighborhood size is illustrated in Figure 3. Since it is nearly impossible to choose an appropriate neighborhood size in advance, we consider different neighborhood sizes. Specifically, we compare respectively the similarity between the two segments around a point whose lengths are within 5%, 10%, 15% and 20% of the total number of points on a streamline. We did not use a fixed number of streamline points for neighborhood size because different streamlines in the same flow field may have very different numbers of points, and our experiment showed that our approach works well on our test cases.

We consider the following metrics for comparing neighboring segments: (1) velocity direction entropy ratio, (2) tortuosity ratio, (3) curvature and torsion histogram difference, and (4) volume ratio of minimum bounding ellipsoids. In the following, we will explain each of these metrics in detail.

3.2.1. Velocity direction entropy

Xu et al. [26] showed that information theory can be applied to effectively capture important flow features in a vector field. Instead of measuring the velocity direction entropy for regions in a vector field [26], we compute the entropy for a streamline segment as an indicator of its complexity. The same idea has been used in Li et al. [18] for streamline similarity comparison. Intuitively, the more complicated a streamline is, the higher its velocity direction entropy value is. To compute an entropy value for a streamline segment, we need to quantize the 3D vector space into a certain number of bins, count how many velocity vectors fall into each bin, and compute the entropy by its definition:

$$H(x) = -\sum_{i=1}^{n} p_i \log p_i \tag{1}$$

where n is the number of bins and p_i is the fraction of the velocity vectors that fall into bin i. For example, for a straight line segment, its velocity direction entropy is zero since all the velocity vectors fall into the same bin, thus making a certain p_i equal to one and the rest equal to zero. To quantize the 3D vector space, we leverage a sphere partition algorithm [27] and choose empirically the number of bins to be 50 (same as [18]). We found that increasing the number of bins is not always a good idea because it will make a relatively simple-looking streamline have a big velocity direction entropy because vectors pointing in similar directions fall into different bins. After the velocity direction entropy is computed for each of the two neighboring segments, we compute the ratio of the smaller entropy value to the larger one. The ratio will be incorporated into the feature vector of the point which separates the two neighboring segments.

3.2.2. Tortuosity

Tortuosity measures the degree of deviation from a straight line for a streamline segment. Similar to velocity direction entropy, it is also a measure of streamline complexity. However, this metric is able to distinguish two dissimilar segments which have similar velocity direction entropy values (Figure 4), and has been used for streamline similarity comparison [16, 18]. The tortuosity of a streamline segment is defined as:

$$T(x) = \frac{\alpha(x)}{\|x_s - x_e\|} \tag{2}$$

where $\alpha(x)$ measures the arclength of a segment and the denominator is the Euclidean distance between the start and end points of a streamline segment. We compute the tortuosity ratio in the same way as velocity direction entropy ratio, thus making the ratio between zero and one.

3.2.3. Curvature and torsion histogram

A curvature and torsion histogram of a streamline segment reveals its shape characteristics since curvature and torsion are two fundamental properties of a curve [28]. Thus, they can provide the information missing from the above two metrics (i.e., entropy and tortuosity). For example, two streamlines of
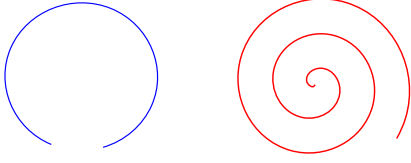
Figure 4: Tortuosity vs. velocity direction entropy: both curves have similar velocity direction entropies because their tangent vectors almost point in every direction in a 2D space. However, the red one looks more complicated than the blue one and has a higher tortuosity value.

very different shapes may have similar velocity direction entropy values.

We compute discrete curvature and torsion as described in [29]. An orthonormal frame X_i, Y_i, Z_i is defined at each streamline point p_{i+1} by

$$X_i = a_i, \ Y_i = \frac{a_{i+1} - (a_{i+1} \cdot a_i)a_i}{\|a_{i+1} - (a_{i+1} \cdot a_i)a_i\|}, \ Z_i = X_i \times Y_i$$

where a_i is the vector from p_i to p_{i+1} and p_i are the points along a streamline.

The discrete curvature $\kappa_i \in [0, \pi)$ at each point p_i and the discrete torsion $\tau_i \in (-\pi, \pi]$ for each segment $p_i p_{i+1}$ are given by the following formulas:

$$\kappa_i = \cos^{-1}(X_i \cdot X_{i-1}),$$
$$\tau_i = \begin{cases} \cos^{-1}(Z_{i-1} \cdot Z_i) & \text{if } Z_{i-1} \cdot a_{i+1} \geq 0 \\ -\cos^{-1}(Z_{i-1} \cdot Z_i) & \text{if } Z_{i-1} \cdot a_{i+1} \leq 0 \end{cases}$$

Note that although curvature and torsion at each point can also be computed via interpolation from the underlying velocity field [30], we choose not to do so because the training examples from different flow fields can be used without scaling if curvature and torsion are computed in the above way. This is because that the best accuracy of SVM can be achieved if the same components across different training examples are in the same unit [31]. Currently, the training examples all come from the same data set. In the future, we may extend our approach to consider training examples from multiple data sets.

In order to obtain a histogram describing the shape characteristics of a streamline, we first compute curvature and torsion histograms respectively, and then concatenate the two histograms into a single 1D histogram. We empirically set the number of bins for curvature histograms to 20 and that for torsion histograms to 40 (i.e., $\pi/20 = 9°$/bin), which is enough for the purpose of similarity comparison. This is because even though the numbers of bins are increased to 80 and 160 for curvature and torsion histograms respectively, we did not see a noticeable difference regarding the clustering results later (refer to Section 3.3.1). Finally, we normalize the 1D histogram to make it scale-invariant.

We measure the distance between two 1D histograms using the earth mover's distance (EMD) [32], which is a cross-bin distance function. As Lu et al. [17] pointed out, considering cross-bin relationship leads to a better accuracy of histogram similarity comparison. Although many approaches have been proposed to reduce the computation cost of EMD such as [33],

our experiment showed that the L_1-distance between two 1D cumulative histograms is a fast yet effective way of approximating EMD [34]:

$$d(P, Q) = \sum_{i=1}^{n} |P_{cdf}(i) - Q_{cdf}(i)|$$

where P_{cdf} and Q_{cdf} are the cumulative distribution functions of the two normalized histograms.

3.2.4. Volume ratio of minimum bounding ellipsoids

For a point p on a streamline and its two neighboring segments s_1 and s_2, consider the following three minimum bounding ellipsoids: $E_{s_1 \cup s_2}$, E_{s_1} and E_{s_2}. The ellipsoids enclose, respectively, all the points which belong to $s_1 \cup s_2$, s_1, and s_2. For p to be a segmentation point, we observe that the following ratio should be small (e.g., less than 1.0):

$$\frac{V(E_{s_1}) + V(E_{s_2})}{V(E_{s_1 \cup s_2})} \tag{3}$$

where V measures the volume of an ellipsoid.

For example, in Figure 5 (a), let p be the red point and s_1 (left) and s_2 (right) be its neighboring segments. Since s_1 is close to a straight line, its bounding ellipsoid has a volume close to zero. However, the bounding ellipsoid for $s_1 \cup s_2$ is much larger than that for s_1, thus making the above ratio small. So intuitively, this heuristic can help separate two neighboring segments with significant complexity difference. It is also a very good indicator for separating 3D features when there is an abrupt change in torsion even though those features have similar complexity. For streamlines which do not have distinct features, this ratio can be larger (e.g., close to 1.0). For example, the ratio is always 1.0 no matter where we separate a straight line, assuming that $\frac{0}{0} = 1.0$. Another example is a helix which can be considered as a complete feature by itself. No matter where we segment the helix, the resulting ratio will always be close to 1.0. On the other hand, let us look at Figure 5 (b) for an example which may require segmentation. The streamline has a lower part which swirls almost in the same plane, and an upper part which looks like a helix. Again, the bounding ellipsoid for the lower part degenerates into an ellipse, thus having a volume of zero. The bounding ellipsoid for the whole streamline is much larger than the one only enclosing the upper part. Section 4.2 will discuss in detail the impact of this heuristic on the classifier performance and final segmentation results.

We use the algorithm proposed by Kumar et al. [35] to compute minimum volume bounding ellipsoids. At first sight, principle component analysis (PCA) [36] seems to be a good way of computing approximate bounding ellipsoids. However, our experiment showed that PCA does not perform as well as [35] in our scenarios. There are of course other bounding shapes such as cubes or spheres, but ellipsoids can bound a streamline more compactly than other shapes.

3.3. Training examples collection

We aim at achieving the best training accuracy with minimal user input. To this end, we automatically pick some representative streamlines for users to segment (Section 3.3.1) and
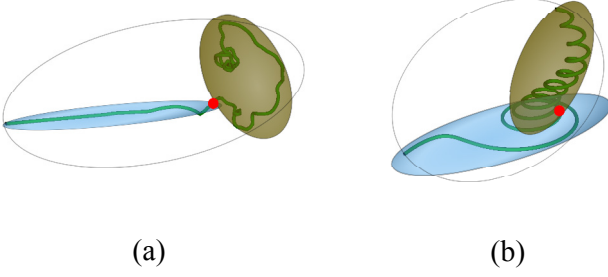
4

(a)   (b)

Figure 5: The blue and the brown ellipsoids are the minimum volume ellipsoids bounding the streamline segments on two sides of the red point. The minimum volume bounding ellipsoid for the whole streamline is shown as white ellipsoids.

(a) ε = 0.5   (b) ε = 1.0   (c) ε = 1.5

Figure 7: A streamline with 147 points is simplified with different Fréchet error ϵ (points left after simplification are shown in red): (a) $\epsilon = 0.5$, 25 points left (b) *epsilon* = 1.0, 18 points left (c) *epsilon* = 1.5, 15 points left.

only require users to specify positive training examples (Section 3.3.2).

3.3.1. *Automatically picking streamlines for training*

After tracing the streamlines using the strategy of random seed placement, we cluster them based on their similarity such that users only need to choose from the representative streamlines for training. In order to make the clustering fully automatic without users having to specify the number of clusters, we use *affinity propagation* [37] for clustering.

Unlike k-means 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 paired similarity values as input, 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 equally possible to be cluster centers. As suggested in [37], a smaller preference value produces a smaller number of clusters.

We construct a 1D histogram for each streamline, and compute the pairwise distances as described in Section 3.2.3. Note that the similarity values are the negative of the approximate EMD. The preference value is chosen to be the median of all the distance values. Figure 6 shows an example of our clustering results of the tornado data set (refer to Section 4.1).

Note that we are not aiming at achieving the "best" clustering results in this step because users are allowed to provide more training examples and re-train our segmentation point classifier later.

3.3.2. *Generating training examples*

Given a streamline, users need to pick the points where they want to segment it. For each segmentation point, we compute the velocity direction entropy ratio, tortuosity ratio, histogram difference, and volume ratio of minimum bounding ellipsoids using different neighborhood sizes (Section 3.2), and these values consist of the feature vector for a positive training example. For points not picked by users, they are considered as non-segm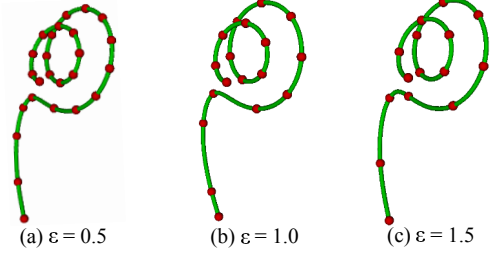entation points. The same features will be computed for each non-segmentation point and consist of the feature vector for a negative training example.

Instead of presenting all the points on a streamline to users, we apply a curve simplification algorithm [38] to reduce the number of candidates for them to choose segmentation points from. There are a couple of reasons for doing curve simplification:

- Ease training example collection. It is difficult for users to pick out a segmentation point if the points are too close to each other. Many points are generated during streamline tracing in order for streamlines to have better visual quality.

- Reduce noisy and redundant training examples. Redundant training examples are generated when nearby points are chosen as segmentation points because these points have similar feature vectors. Ambiguity could happen during the training phase if a nearby point of a segmentation point is chosen to be a non-segmentation point.

- Reduce computation cost. With fewer points on a streamline, we have fewer points to test for segmentation points. Also the cost of training will be reduced since we have less training examples.

The curve simplification algorithm [38] approximates a polygonal curve P under the Fréchet error [38] by another polygonal curve P' whose vertices are a subset of the vertices of P. Figure 7 compares the simplification results using different Fréchet errors. In our experiment, we choose the default Fréchet error to be 1.0 because: (1) this value is conservative enough (i.e., no over-simplification) such that all the points we want to pick as segmentation points are in the simplified streamlines, and (2) reducing its value will result in more points after simplification, which in turn generates more redundant negative training examples because the points used as negative training examples are too close to each other (see next section for details on how negative training examples are generated). We also make the Fréchet error adjustable by users in case the default value does not work well for them.

Since determining where to segment a streamline is often not a clear-cut decision, we allow users to specify an interval on a streamline such that any point in that interval could be a segmentation point. In other words, users may pick a few
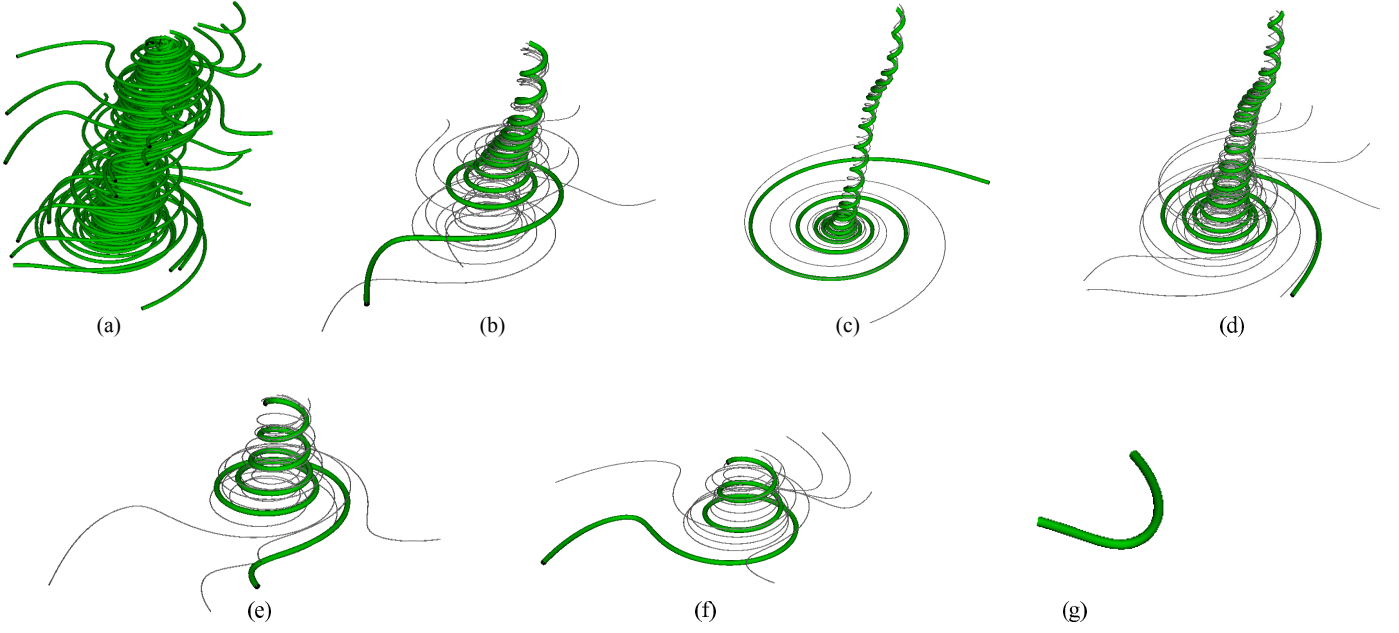
5

Figure 6: Streamline clusters and their representatives: the input streamlines of the tornado data set are shown in (a). After applying affinity propagation, six clusters ((b)-(g)) are obtained and cluster representatives are shown in green.

nearby points to indicate where segmentation occurs. This also helps alleviate the problem of highly imbalanced training examples (refer to Section 3.4) since more positive examples are generated.

3.4. Training

We leverage libSVM [39] to train a classifier for segmentation points. The biggest challenge is imbalanced training examples: we have far more negative examples than positive ones. With such training data, we would get a classifier with a high false negative rate if the standard SVM technique is used. To address this issue, libSVM allows users to set different penalties for positive and negative training examples so that a less biased decision boundary can be obtained. In other words, we could set a very high penalty for misclassifying the positive examples and a small penalty for misclassifying the negative examples. However, our experiment showed that this method does not work very well for our application.

Instead, we opt for a method described in [40] to handle highly imbalanced training data for SVM classifiers. It performs undersampling repeatedly until the classification performance cannot be improved. For highly imbalanced data, it is not a good idea to measure classification performance by the ratio of the number of correctly classified examples to the total number of examples. For example, assume that we have three positive examples and 97 negative examples and a classifier which classifies all the examples as negative. Although the classification accuracy is 97%, it is apparently not a good classifier. Therefore, we use a measure called "area under receiver operating characteristics (ROC) curve" (AUC) [41] to measure classifier performance. An ROC curve is a two-dimensional graph in which true positive rate is plotted on the Y axis and false positive rate is plotted on the X axis. In other words,

instead of just focusing on the number of correctly classified examples, an ROC curve depicts the relative tradeoffs between benefits (true positives) and costs (false positives). To make it easier to compare classifiers, ROC performance can be reduced to a single scalar value representing the expected performance. A common method is to calculate the area under the ROC curve, abbreviated AUC. The value of AUC is always between 0 and 1. Any realistic classifier should have an AUC greater than 0.5.

The pseudo-code for training our segmentation point classifier is given in Algorithm 1. Initially, *aggregation* is initialized to only contain positive training examples (Line 2). Then the algorithm repeats the following steps until classifier performance cannot be improved: (1) train a linear SVM on the input *examples* which initially is the highly imbalanced training data (Line 5); (2) remove the negative support vectors found by the linear SVM from *examples* and replace the negative examples in *aggregation* with them (Lines 6-7); (3) train an SVM classifier using RBF kernel (see Section 3.1) on *aggregation* and check if performance is improved (Lines 8-13).

We follow the suggestions in [31] to perform *grid-search* and *cross-validation* for best classifier performance. Both linear and RBF SVM require a penalty parameter C, and RBF SVM also requires another parameter σ. The best values of these parameters for a given problem are not known beforehand. Hence, some kind of model selection (parameter search) must be done. The goal is to identify good parameter values so that the classifier can accurately predict unknown data. A common strategy is cross-validation. In v−fold cross-validation, training data is divided into v subsets of equal size. Sequentially one subset is tested using the classifier trained on the remaining $v - 1$ subsets. A grid-search is performed on C and/or σ using cross-validation. We perform the grid-search on the

6

Algorithm 1 Classifier training procedure

```
 1: procedure TRAINSEGPOINTCLASSIFER(examples)
 2:     aggregation ← all the positive training examples
 3:     bestAUC ← 0
 4:     while true do
 5:         linearModel ← LINEARSVM(examples)
 6:         aggregation.nSVs ← linearModel.nSVs
 7:         examples.ERASE(linearModel.nSVs)
 8:         segPointClassifier ←RBFSVM(aggregation)
 9:         auc ←COMPUTEAUC(segPointClassifier)
10:         if auc ≤ bestAUC then
11:             break
12:         else
13:             bestAUC ← auc
```

exponentially growing sequences of C and σ as suggested by [31]: $C = \{2^{-5}, 2^{-3}, \cdots, 2^{15}\}$, and $\sigma = \{2^{-15}, 2^{-13}, \cdots, 2^3\}$.

3.5. Segmentation and post-processing

Streamline segmentation is straightforward once the above classifier is obtained. For each streamline, we check for each point on the simplified streamline whether it is a segmentation point. It is possible that several nearby points are classified as segmentation points (Figure 8 (a)). Therefore, we need to group them and pick one of them as a segmentation point.

The goal is to group segmentation points which are close to each other in terms of arc length. In order to achieve this grouping, we use half of the mean of arc lengths between the segmentation points as threshold T_S and group the segmentation points for which the arc length is less than T_S. The same strategy was also adopted by [42] to group salient points of a 3D mesh.

Formally, assume that $S = \{s_i | 1 \le i \le N_S\}$ is the set of segmentation points found for a streamline, then the threshold T_S is defined as:

$$T_S = \frac{\sum_{i=1}^{N_S-1} \sum_{j=i+1}^{N_S} \alpha(s_i, s_j)}{N_S(N_S - 1)}$$

where $\alpha(s_i, s_j)$ measures the arclength between segmentation points s_i and s_j.

Note that the first and the last point of a streamline have to be included when computing the threshold T_S since they should be considered as segmentation points as well (i.e., the start point of the first segment and the end point of the last segment).

A group C of segmentation points is defined as:

$$C = \{s_i \in S : \forall s_j \in C, \alpha(s_i, s_j) \le T_S\}$$

The final segmentation point within each group is the one which has the smallest ratio of minimum bounding ellipsoids. In other words, for each segmentation point s_i in a group, we compute the minimum bounding ellipsoids (refer to Section 3.2.4) of its left segment $s_p s_i$, its right segment $s_i s_n$, and both its left and right segments. The segmentation point in this group is the one which gives the smallest ratio as computed by
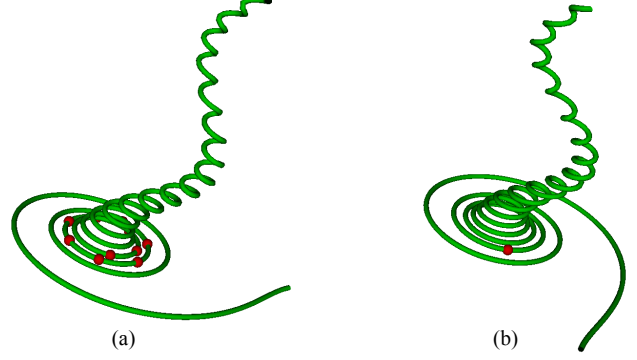

Figure 8: Remove redundant segmentation points: (a) nearby points (in red) are detected as segmentation points by our trained classifier. (b) only one segmentation point is left after post-processing.

Equation 3. The point s_p (s_n) is an arbitrary point from the previous (next) group along the streamline (since points in the same group are close to each other, picking an arbitrary point will not affect segment shape much). Figure 8 shows an example result of our grouping algorithm.

4. Results and Discussion

Our experiment was performed on a laptop with an Intel Core i5-3360M CPU running at 2.8GHz, 8GB main memory and an AMD FirePro M2000 graphics card. Only a single CPU thread is used for all the computations.

4.1. Partial flow feature exploration

We validated our segmentation algorithm by clustering the segments based on their similarities. The similarity between two segments is measured using the method introduced in Section 3.2.3, and then affinity propagation is applied to obtain the clusters. Three steady flow data sets (Table 1) are used in our experiment. The tornado data set is procedurally generated by software. The five critical points data set is a synthesized flow field consisting of two spirals, two saddles, and one source. Finally, the solar plume data set is from a simulation of down-flowing solar plumes for studying the heat, momentum and magnetic field of the sun. For each data set, the pool of streamlines used during the training stage and the one used for segmentation were traced *separately* in order to test the generalization ability of the trained classifier. In the following, we show the streamlines used for training and the streamline segment clusters for each data set. The user-picked segmentation points are highlighted in blue, which are used as positive training examples. The remaining points in red are used as negative training examples.

Case Study 1 – Tornado Data Set (Figure 9). Five (t_1-t_5) out of 30 traced streamlines were manually segmented by us. We did not segment the streamline in t_4 because it does not contain any interesting partial feature (e.g., spirals) which we would like to extract. The clustering results (s_1-s_7) show that user-defined features were extracted successfully. For example, the segments in cluster s_1 correspond to the bottom swirl in t_1

whose points have a torsion close to zero. The segments in cluster s_2 look very similar to the one from t_5 which has an inflection point. However, we do notice that a few streamlines from the cluster s_7 failed to be segmented as desired (circled in purple).

Case Study 2 – Five Critical Points Data Set (Figure 10). Seven (t_1 - t_7) out of 60 traced streamlines were manually segmented. A total of eight clusters (s_1-s_8) were generated after segmenting each streamline and clustering the resulting segments. Two swirls (s_4 and s_5) which are contained in the original data set but occluded by other surrounding streamlines are successfully extracted. They were put into two different clusters because of their shape difference. The source is also revealed (s_3 and s_6). The segmentation took less time (Table 1) for this data set because each streamline has a smaller number of points compared with other data sets.

Case Study 3 – Solar Plume Data Set (Figure 11). Some streamlines from this data set are even more difficult for us to determine the "best" segmentation. So we manually segmented 14 streamlines in order to extract interesting features instead of determining the best segmentations. After segmentation and clustering, we had 13 clusters and only eight of them are shown in Figure 11 due to space limit. Readers can refer to the accompanied video for complete results. It can be seen that the features specified by us were successfully extracted. For instance, the features similar to the spirals specified by users in t_1 and t_4 appear in clusters s_1, s_4 and s_6. Furthermore, the cluster s_1 contains the features similar to the letter "J" shape as shown in t_9, t_{10} and t_{11}. The fact that some spirals appear in cluster s_1 reveals a limitation of our similarity measure. As seen from the figure, these spirals were successfully separated from the remaining streamlines, which suggests that our segmentation algorithm works well. Although segments in clusters s_3 to s_6 are perceptually similar, they are unfortunately separated into different clusters because of the similarity measure and clustering algorithm. The same happened for clusters s_7 and s_8.

4.2. Feature selection

In the following, we show that the metrics currently incorporated into feature vectors are relevant, and in particular, the volume ratio of minimum enclosing ellipsoids greatly improves the classifier performance. Denote the four metrics velocity direction entropy ratio, tortuosity ratio, curvature and torsion histogram difference, and minimum bounding ellipsoid volume ratio by $M1$, M_2, M_3, and M_4, respectively. Also let $G_1 = \{M_1, M_2, M_3, M_4\}$, $G_2 = \{M_1, M_2, M_3\}$ and $G_3 = \{M_4\}$.

For each data set, we manually segmented the *same* set of streamlines. Three different sets of training examples were generated using G_1, G_2, and G_3, respectively. Then we trained one classifier from each set of training examples. Finally, we compare the results *before post-processing* of segmenting the same set of streamlines using the three classifiers. By comparing between the segmentation results obtained from using G_1 and G_2 respectively, we can intuitively see how much contribution M_4 makes to the classifier performance. However, using M_4 alone is not sufficient to get an accurate classifier, which can be observed from the segmentation results using G_3.

As Table 2 shows, classifiers trained using G_1 generally give the best segmentation results and require the least amount of training time. Segmentation results are greatly improved when the volume ratio of minimum enclosing ellipsoids is incorporated into feature vectors (compare G_1 and G_2 columns). However, using it alone (G_3) does not give satisfactory segmentation results and also requires a much longer training time.

4.3. Parameters

There are two parameters which will affect the final segmentation results: (1) the number of different neighborhood sizes used during multiscale feature computation (Section 3.2), and (2) the distance threshold used to group nearby segmentation points (Section 3.5). The second parameter is computed based on the segmentation points already found by a classifier, hence, users should not be allowed to adjust its value. In the following, we discuss how the first parameter will affect the performance of a classifier in terms of AUC and training time.

In this experiment, we use the tornado data set to generate 36 positive and 298 negative examples. We consider the following neighborhood sizes: 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40%, 45%, 50% of the total streamline points. (Since at most 500 points are generated when tracing streamlines for our data sets, two neighboring neighborhood sizes differ by 25 points at most. Note that in our experiment a streamline segment consisting of 25 points is usually too tiny to be visually considered as a standalone segment.) We combine different neighborhoods in the following way when computing multiscale feature vectors: starting from a certain neighborhood size, n consecutive neighborhoods up to 50% of the total number of streamline points are used. For example, if we start at 5% and $n = 4$, it means that the neighborhoods whose sizes are 5%, 10%, 15% and 20% of total streamline points will be used for computing feature vectors. Table 3 lists the statistics of AUC and training time for different neighborhood combinations.

Table 3 shows that using only one or two neighborhood sizes (e.g., columns 1 and 2) can generally give a high AUC value but with a long training time. We found that the long training time was due to the fact that libSVM [39] took a long time to converge and in many cases it even reported the maximum number of iterations was reached. Moreover, the segmentation results were very bad when only one or two neighborhood sizes were used. This suggests that high AUC values in these cases were due to over-fitting, so the classifiers did not generalize well.

As more neighborhoods were considered, the value of AUC generally increased, so did the training time (e.g., rows 5% and 10%). The training time generally increased by a few seconds. We checked the segmentation results obtained with the starting neighborhood size being 5% and 10 consecutive neighborhoods being used, and found that the segmentation results indeed were very good. However, segmentation itself took a much longer time because more computation is required.

The values in Table 3 also suggest that the starting neighborhood size should not be set too large because the AUC values from the row 20% and below are generally not as good as that from the rows above.
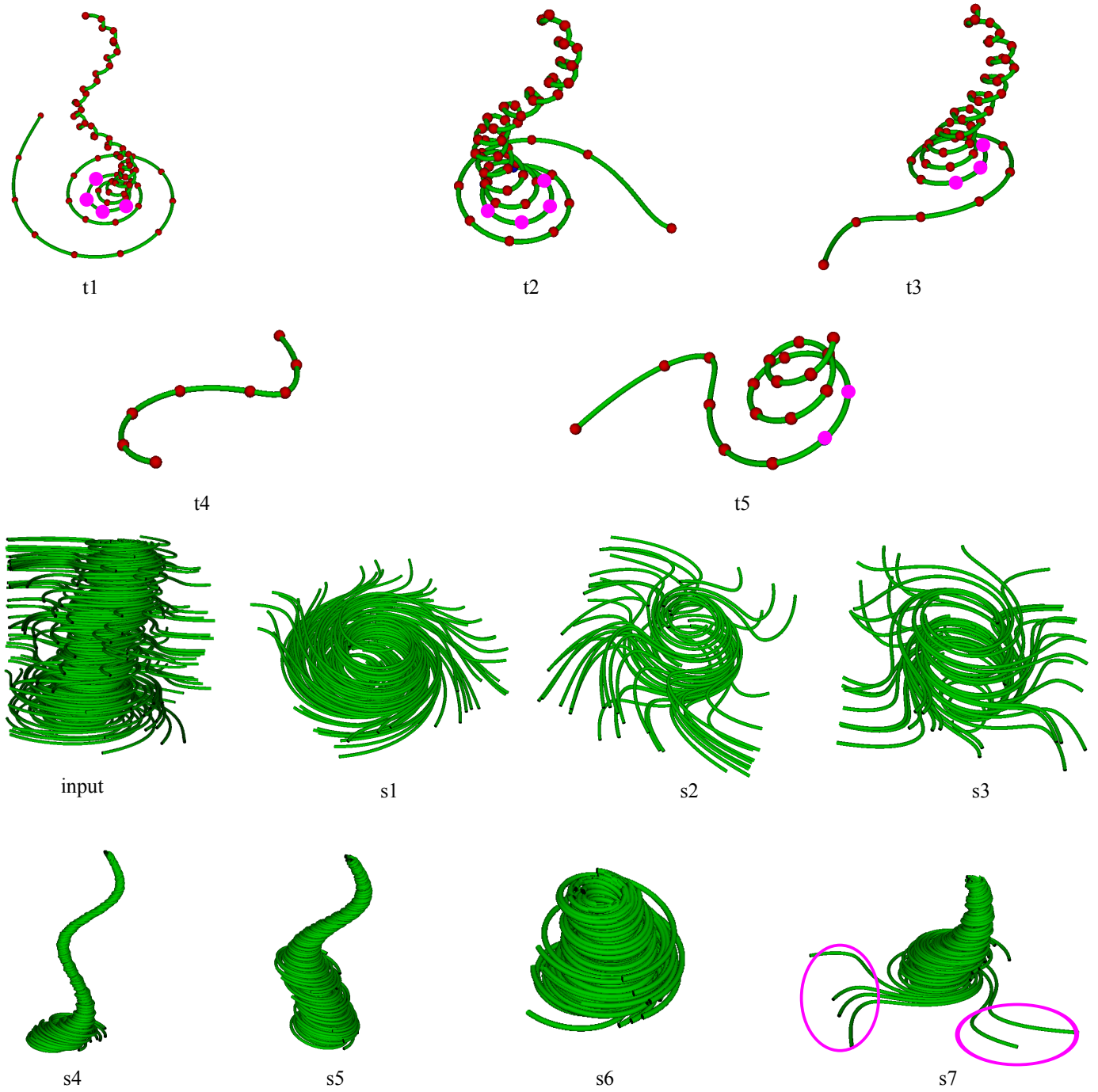
8

Figure 9: Five streamlines of the tornado data set were manually segmented (t_1-t_5) to train the classifier. The user-picked segmentation points are highlighted in purple, which are used to generate positive training examples. The remaining points in red are used to generate negative training examples. The segmentation was performed on the streamlines which were traced separately from the training streamlines. Seven (s_1-s_7) clusters of streamline segments were generated. Note that users can specify a few nearby points as possible segmentation points (e.g., t_2)

Figure 10: Seven streamlines of the five critical points data set were manually segmented (t_1-t_5) to train the classifier. Eight (s_1-s_8) clusters of streamline segments were generated.
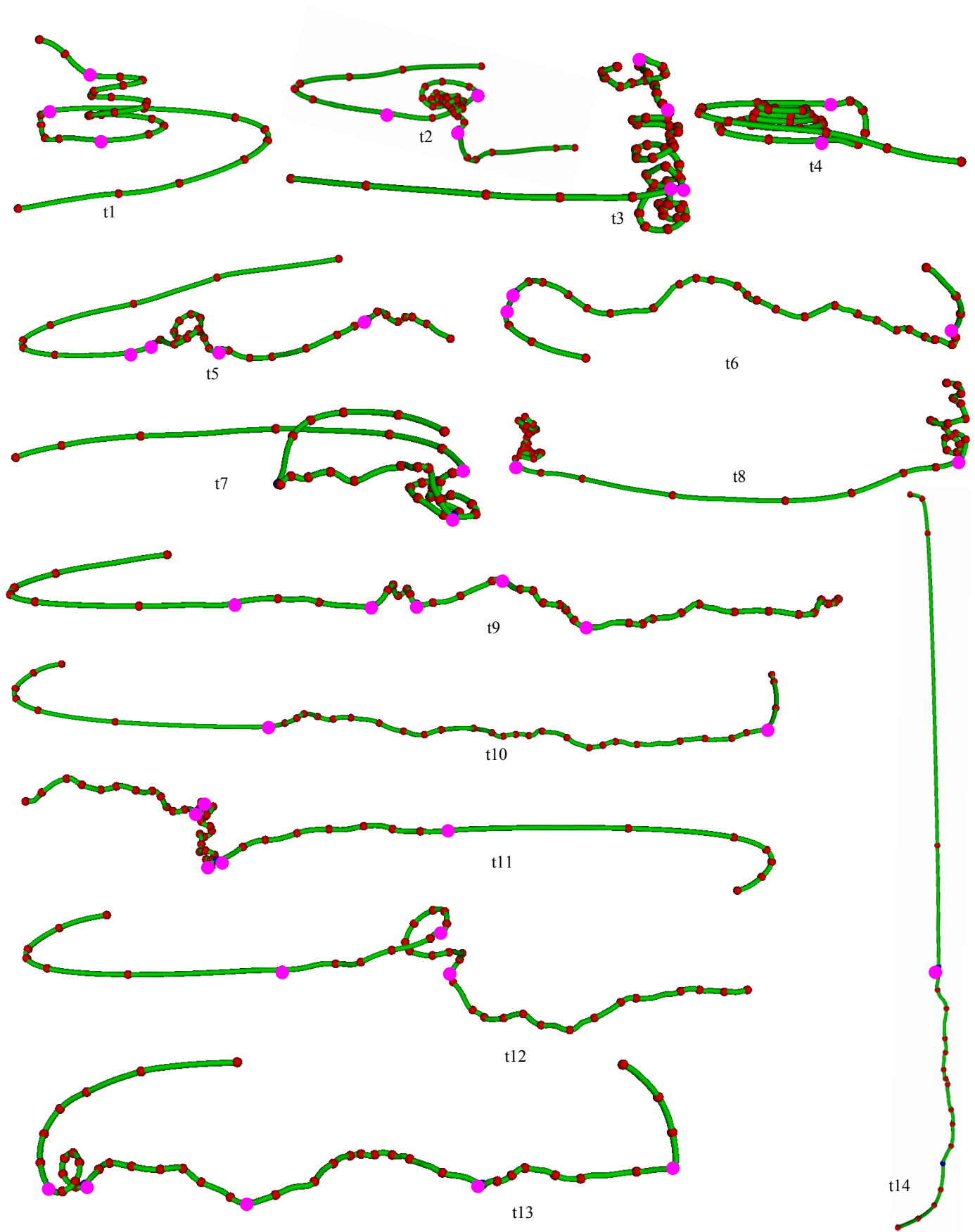
10

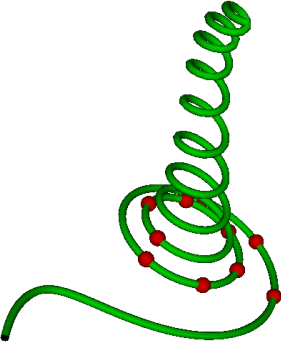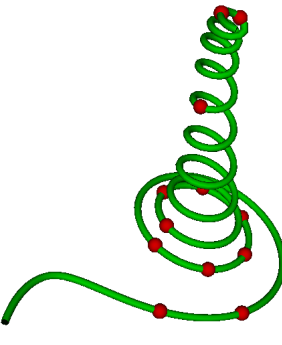Figure 11: 14 streamlines of the solar plume data set (t_1-t_{14}) were manually segmented for training.

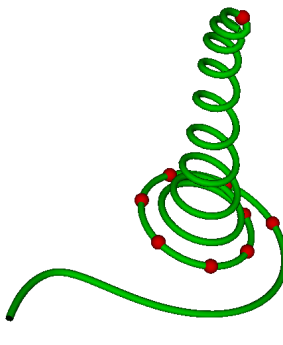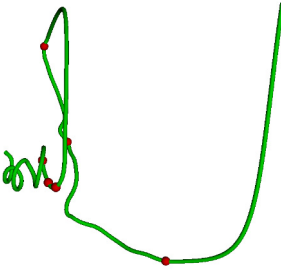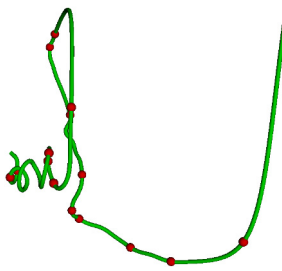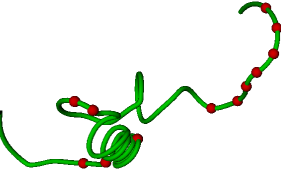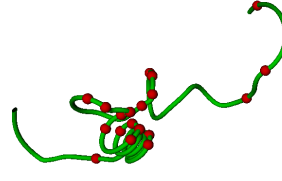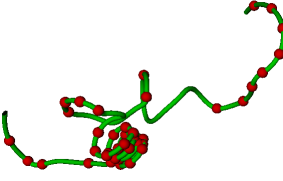Figure 11 (cont.): Eight (s_1-s_8) out of 14 clusters of streamline segments of the solar plume data set are shown here. Notice how the interesting features such as spirals and turbulent features are successfully extracted.

12

Table 1: The three flow data sets. The timing results are in seconds.

data set	dimension	training					segmentation		
		# lines	# seg. lines	# pos/neg examples	training time	AUC	# lines	simp. time	seg. time
tornado	$64 \times 64 \times 64$	30	5	22/157	3.54	0.99	150	1.99	35.76
five critical pts	$51 \times 51 \times 51$	60	7	10/52	3.80	0.90	150	0.60	2.74
solar plume	$126 \times 126 \times 512$	80	14	70/414	59.01	0.97	200	5.18	59.32

Table 2: Segmentation results without post-processing using the classifiers trained with different types of feature vectors $G_1 = \{M_1, M_2, M_3, M_4\}$, $G_2 = \{M_1, M_2, M_3\}$ and $G_3 = \{M_4\}$, where $M1$, M_2, M_3, and M_4 are velocity direction entropy ratio, tortuosity ratio, curvature and torsion histogram difference, and minimum bounding ellipsoid volume ratio, respectively.

G_1	G_2	G_3	# pos/neg examples & training times

- data set : tornado
- 28/195
- G_1 : 6.59s
- G_2 : 8.23s
- G_3 : 30.69s

- data set: crayfish
- 25/314
- G_1 : 20.22
- G_2 : 43.32
- G_3 : 259.44

- data set: crayfish
- 25/314
- G_1 : 20.22
- G_2 : 43.32
- G_3 : 259.44

13

Similar findings were also obtained for other data sets. Therefore, we empirically choose the following neighborhood sizes, $\{5\%, 10\%, 15\%, 20\%\}$, when computing multiscale feature vectors in order to get a balance between good classifier performance and short segmentation time.

5. Comparison

In this section, we will first compare our supervised streamline segmentation method with the other two segmentation algorithms [11, 17], and then compare the partial flow features extracted by our segmentation algorithm and that by Tao et al. [10].

Lu et al. [17] proposed an iterative top-down segmentation algorithm. Their algorithm recursively segments a streamline into two most dissimilar segments until either the dissimilarity is below a certain threshold t_s or the number of points contained in current segment is less than t_l. We implemented their algorithm using the EMD distance between the two curvature and torsion histograms (see Section 3.2.3) as the dissimilarity measure.

The major issue of their approach is that these two parameters are not intuitive for users to adjust manually, which is illustrated by Figure 12 (leftmost column). The streamlines in rows (a) and (b) are from the tornado data set, and that in rows (c) and (d) are from the solar plume data set. Setting t_l and t_s to 50 and 1.0 resulted in over-segmentation for the streamlines in rows (a) and (b). Increasing t_l to 100 avoided generating a small segment for the streamline in row (a), but also left the streamline in row (b) not segmented at all. The over-segmentation problem for the streamline in row (c) cannot be solved easily by increasing t_l. By carefully adjusting the value of t_s from 0.95 to 1.0, the part which looks like a spiral was separated out successfully. However, the U-shape at the bottom failed to be in its own segment. As seen in the rightmost column, our method does not suffer from these problems. Finally, it is hard to say which segmentation is better for the streamline in row (d) because the extra segmentation point from Lu's method occurred in a turbulent region. Both segmentations look acceptable to us. In conclusion, this approach is not good for general streamline segmentation but for the cases where a streamline needs to be divided into segments for further processing such as [17].

The streamline segmentation algorithm by Wang et al. [11] is a bottom-up approach. A streamline is first split into *minimal segments* which are bounded by points of absolute local curvature minima. The minimal segments are then merged based on a two-phase compatibility test. First, two neighboring segments are mergeable if they have similar average orientations, i.e., if the angle between the two segments' average binormal directions is less than t_α. Second, a segment with a low total curvature less than t_κ is merged with its two neighboring segments. The merging algorithm iteratively processes segments based on a priority queue that is ordered by the total segment curvature. To implement this algorithm, we measure the binormal direction of a segment between two consecutive points as the cross product of the velocity vectors at those points.

Wang et al. [11] claimed that their segmentation algorithm meets the following three requirements: (1) a segmentation should be *feature preserving* in that important features should be preserved, (2) a segment should be *distinct* enough to describe a complete feature, and (3) streamlines describing similar flow features should be segmented *consistently*. However, we found that their algorithm cannot guarantee to meet these requirements. For example, their algorithm produced over-segmentation for the streamline in row (a) because some minimal segments have a relatively large total curvature. After changing the value of t_κ from 1.0 to 1.8, the problem was alleviated but the final segmentation still does not look natural. The segmentation result in row (b) looks acceptable. The streamline in row (c) was also over-segmented with $t_\kappa = 1.4$, however, increasing its value to 1.5 failed to preserve the spiral feature. Finally, the over-segmentation of the streamline in row (d) cannot be easily solved due to the turbulent nature of the enlarged area: the neighboring minimal segments have very different average binormal directions and also a large total curvature. Therefore, this approach may work well for flow fields which do not have many turbulent regions (as illustrated in [11]), but it is not a good choice to segment turbulent streamlines.

Since the above two methods cannot segment individual streamlines into different features satisfactorily for our data sets, we now compare our method with FlowString ([10]) on flow feature extraction. Tao et al. [10] represented each streamline as a string and the substrings which appear frequently are considered as interesting flow features. Two parameters, minimum length and minimum frequency, can be adjusted by users to search for frequent substrings. The FlowString library is available at [43], and is used during our comparison.

Figure 13 shows the features extracted by FlowString for tornado (a-f) and solar plume (g-l) data sets, respectively. For each of the two data sets, the same set of streamlines was traced as the input used in the previous case studies (Section 4.1). It can be seen that FlowString has the following shortcomings compared to our method:

- FlowString may return many similarly-looking patterns. For example, three types of features (corresponding to three different substrings) out of 19 are shown in Figure 13 (a)-(c), which look similar to each other (the remaining features also look similar, and hence are not shown to save space). The features are extracted with minimum frequency and minimum length set to 100 and 3 respectively. For the tornado data set, the features found by our method were clustered into 7 distinguishable groups (Figure 9).

- Users need to adjust the value of minimum frequency (minimum length) to search for the desired features. For instance, the features in Figure 13 (e)-(f) were only available when we decreased the value of minimum frequency from 100 to 50 . For the solar plume data set, only two types of features ((g)-(h)) were extracted when minimum frequency and minimum length were set to 100 and 4 respectively. The spiral features in (j)-(l) did not appear unless the minimum frequency was set to a small value

Table 3: AUCs and training times (in seconds) for the classifiers trained using different combinations of neighborhood sizes. The training was conducted on 36 positive and 298 negative training examples generated from the tornado data set.

| starting neigh. size	# consecutive neighborhoods (AUC/training time)									
	1	2	3	4	5	6	7	8	9	10
5%	0.95/46.0	0.99/30.1	0.93/8.7	0.92/8.7	0.94/12.2	0.97/12.1	0.97/5.8	0.97/5.6	0.96/5.1	1.0/8.1
10%	1.0/69.4	0.98/28.9	0.97/10.8	0.97/6.3	0.98/8.5	0.96/8.5	1.0/11.8	1.0/10.5	1.0/15.1	
15%	1.0/83.1	0.97/36.6	0.92/7.1	0.94/8.6	0.99/10.9	0.97/11.9	0.94/13.8	0.98/12.4		
20%	0.99/171.3	0.96/41.6	0.92/18.8	0.90/23.6	0.93/17.3	0.90/16.6	0.91/14.0			
25%	0.98/162.0	0.959/63.5	0.958/49.8	0.94/47.8	1.0/63.1	0.96/39.4				
30%	0.98/95.6	0.93/44.9	0.91/28.1	0.92/98.1	0.88/64.8					
35%	0.90/49.9	0.909/23.3	0.92/29.2	0.91/34.6						
40%	1.0/160.2	0.90/34.1	0.93/64.0							
45%	0.91/34.5	0.95/58.9								
50%	1.0/93.2									
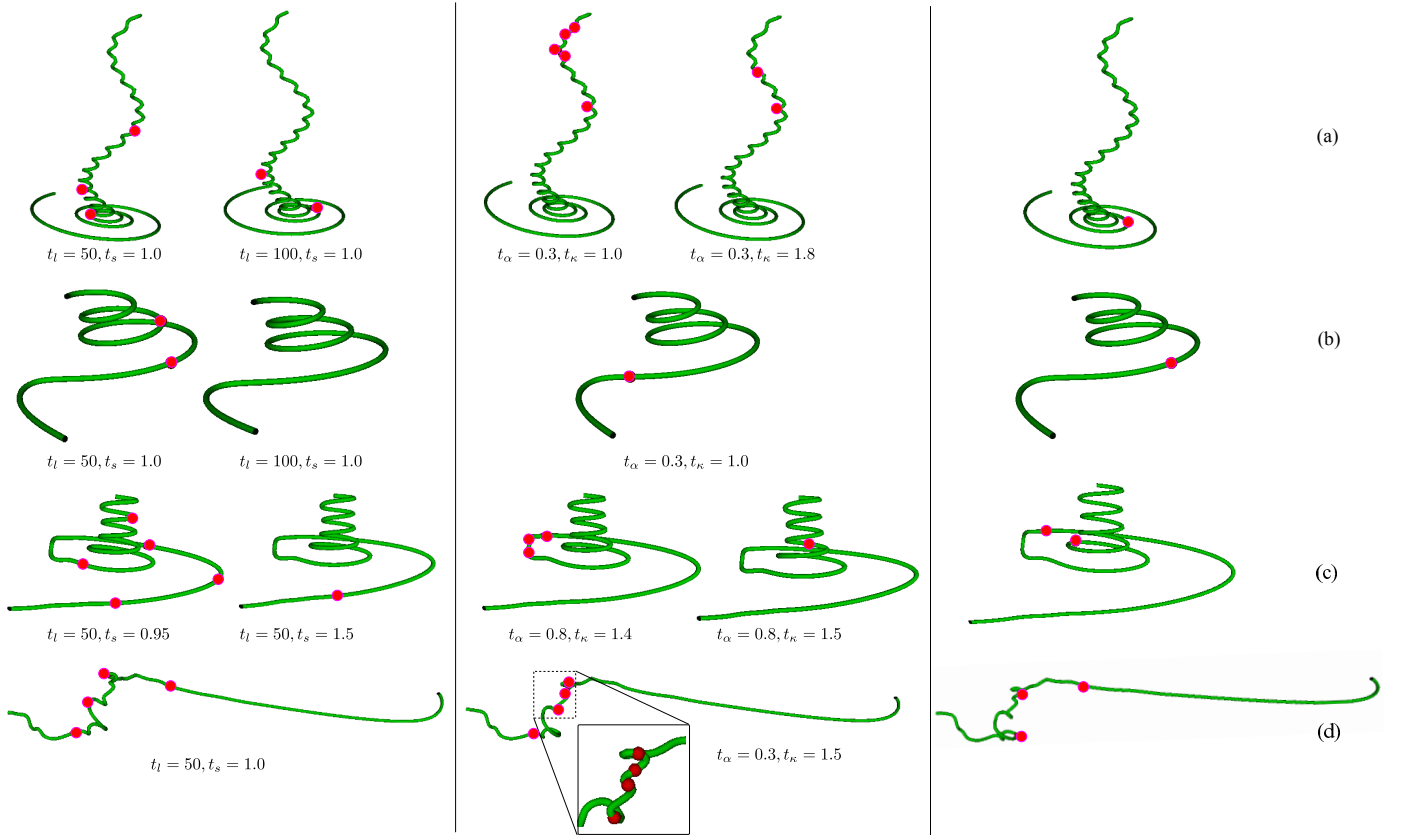
Figure 12: A comparison on streamline segmentation between [11, 17] (first two columns) and our method (last column). The streamlines in row (a) and (b) are from the tornado data set, and those in rows (c) and (d) from the solar plume data set. The segmentation points are highlighted in red.
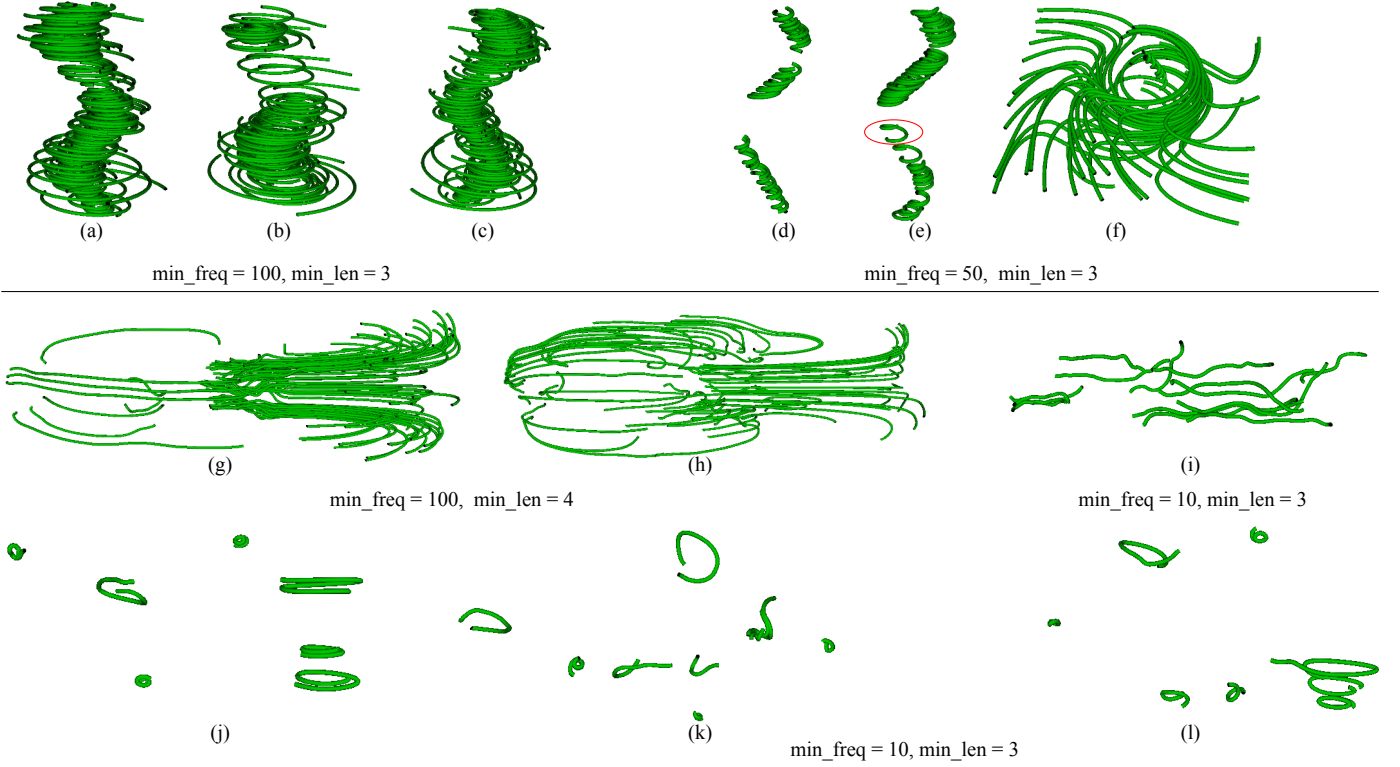
15

Figure 13: The features extracted by FlowString [10] for tornado (a-f) and solar plume (g-l) data sets, respectively.

of 10. In contrast, since our method already segments each streamline into different features, the final segment clusters usually include all the desired features.

- The features matched by FlowString often correspond to incomplete features. This problem is clearly illustrated in Figure 13 (d)-(e), where partial spiral features were extracted (e.g., the spiral in the red circle). The same problem also occurs for the spirals extracted in (j)-(l). However, our method is able to find more complete features (see Figure 9 and 11).

The advantage of FlowString over our method is that it is fully automatic (no user intervention is required) and it does not require computing as many features as our method (although it requires registration computation for each pair of segments).

Finally, we do want to point out a limitation of our method : a classifier trained on one data set cannot be applied to another data set which has very different streamlines. In order to get a classifier which can work across multiple data sets, a central database may be required to store all the training examples similar to [44], and incremental training should be performed.

6. Conclusions and Future Work

In this paper, we propose a novel streamline segmentation algorithm based on user-defined features using machine learning. To the best of our knowledge, this is the first work in flow visualization which leverages supervised training for feature extraction. We also make another contribution by propos-

ing a very effective heuristic for streamline segmentation: the volume ratio of minimum enclosing ellipsoids.

Our algorithm automatically picks a few representative streamlines for users to segment. The user input is then turned into feature vectors, which in turn are trained to obtain a classifier for segmentation points. Streamline segmentation then becomes a process of segmentation point testing via the classifier. Finally, a post-processing step is applied to remove nearby segmentation points found by the classifier.

The results are encouraging, and we point out the following directions for future work:

- Identify more effective metrics besides the ones mentioned in Section 3.2 and incorporate them into feature vectors. It would be interesting to work with experts from human perception or cognitive science to find out the rules which human beings use to segment streamlines, or 3D curves in general.

- Apply this approach to time-dependent data. Segmenting pathlines and clustering similar pathline segments would allow us to better understand features in unsteady flow fields.

- Generate a hierarchical segmentation for each streamline. Our current approach only generates a single segmentation. However, human beings tend to segment a geometric object in a hierarchical manner (e.g., hierarchical mesh segmentation [23]).

- Improve computation speed. All the experiments were

done using a single CPU thread. We would like to leverage CUDA/OpenCL for real-time streamline segmentation.

The implementation details of our approach is available at http://www.nd.edu/~cwang11/streamline-segmentation.html

References

[1] Z. Peng, R. S. Laramee, Higher dimensional vector field visualization: A survey, in: Proceedings of Theory and Practice of Computer Graphics, 2009, pp. 149–163.

[2] R. S. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F. H. Post, D. Weiskopf, The state of the art in flow visualization: Dense and texture-based techniques, Computer Graphics Forum 23 (2) (2004) 203–221.

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

[4] T. Salzbrunn, H. Jänicke, T. Wischgoll, G. Scheuermann, The state of the art in flow visualization: Partition-based techniques., in: Proceedings of Simulation and Visualization Conference, 2008, pp. 75–92.

[5] A. Brambilla, R. Carnecky, R. Peikert, I. Viola, H. Hauser, Illustrative flow visualization: State of the art, trends and challenges, Eurographics State-of-the-Art Reports (2012) 75–94.

[6] M. Edmunds, R. S. Laramee, G. Chen, N. Max, E. Zhang, C. Ware, Surface-based flow visualization, Computers & Graphics 36 (8) (2012) 974–990.

[7] M. Schlemmer, M. Heringer, F. Morr, I. Hotz, M.-H. Bertram, C. Garth, W. Kollmann, B. Hamann, H. Hagen, Moment invariants for the analysis of 2D flow fields, IEEE Transactions on Visualization and Computer Graphics 13 (6) (2007) 1743–1750.

[8] R. Bujack, I. Hotz, G. Scheuermann, E. Hitzer, Moment invariants for 2D flow fields using normalization, in: Proceedings of IEEE Pacific Visualization Symposium, 2014, pp. 41–48.

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

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

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

[12] S. Zhang, S. Correia, D. H. Laidlaw, Identifying white-matter fiber bundles in DTI data using an automated proximity-based fiber-clustering method, IEEE Transactions on Visualization and Computer Graphics 14 (5) (2008) 1044–1053.

[13] J. S. Shimony, A. Z. Snyder, N. Lori, T. Conturo, Automated fuzzy clustering of neuronal pathways in diffusion tensor tracking, in: Proceedings of International Society of Magnetic Resonance in Medicine, Vol. 10, 2002.

[14] H. Yu, C. Wang, C.-K. Shene, J. H. Chen, Hierarchical streamline bundles, IEEE Transactions on Visualization and Computer Graphics 18 (8) (2012) 1353–1367.

[15] I. Corouge, S. Gouttard, G. Gerig, Towards a shape model of white matter fiber bundles using diffusion tensor MRI, in: Proceedings of IEEE International Symposium on Biomedical Imaging: Nano to Macro, 2004, pp. 344–347.

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

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

[18] Y. Li, C. Wang, C.-K. Shene, Streamline similarity analysis using bag-of-features, in: Proceedings of IS&T/SPIE Electronic Imaging, 2013.

[19] T. Schultz, Feature extraction for DW-MRI visualization: The state of the art and beyond, Scientific Visualization: Interactions, Features, Metaphors 2 (2011) 322–345.

[20] T. Günther, C. Rössl, H. Theisel, Opacity optimization for 3D line fields, ACM Transactions on Graphics 32 (4) (2013) 120.

[21] P. L. Rosin, G. A. West, Nonparametric segmentation of curves into various representations, IEEE Transactions on Pattern Analysis and Machine Intelligence 17 (12) (1995) 1140–1153.

[22] M. L. Braunstein, D. D. Hoffman, A. Saidpour, Parts of visual objects: An experimental test of the minima rule, Perception 18 (6) (1989) 817–826.

[23] A. Shamir, A survey on mesh segmentation techniques, Computer Graphics Forum 27 (6) (2008) 1539–1556.

[24] C. M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

[25] B. Scholkopf, A. J. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, MIT press, 2001.

[26] L. Xu, T.-Y. Lee, H.-W. Shen, An information-theoretic framework for flow visualization, IEEE Transactions on Visualization and Computer Graphics 16 (6) (2010) 1216–1224.

[27] P. Leopardi, A partition of the unit sphere into regions of equal area and small diameter, Electronic Transactions on Numerical Analysis 25 (12) (2006) 309–327.

[28] T. Shifrin, Differential geometry: a first course in curves and surfaces, http://www.math.uga.edu/~rscott/teaching/4250_Sp2013/ShifrinDiffGeo.pdf, accessed: 2014-12-30.

[29] D. E. Blair, T. Konno, Discrete torsion and its application for a generalized van der Waerden's theorem, Proceedings of the Japan Academy, Series A, Mathematical Sciences 87 (10) (2011) 209–214.

[30] T. Weinkauf, H. Theisel, Curvature measures of 3D vector fields and their applications, Journal of WSCG 10 (2002) 507–514.

[31] A practical guide to support vector classification, http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf, accessed: 2014-12-30.

[32] Y. Rubner, C. Tomasi, L. J. Guibas, The earth mover's distance as a metric for image retrieval, International Journal of Computer Vision 40 (2) (2000) 99–121.

[33] O. Pele, M. Werman, Fast and robust earth mover's distances, in: Proceedings of IEEE International Conference on Computer Vision, 2009, pp. 460–467.

[34] S. Cohen, L. Guibas, The earth mover's distance under transformation sets, in: Proceedings of IEEE International Conference on Computer Vision, 1999, pp. 1076–1083.

[35] P. Kumar, E. A. Yildirim, Minimum-volume enclosing ellipsoids and core sets, Journal of Optimization Theory and Applications 126 (1) (2005) 1–21.

[36] I. Jolliffe, Principal Component Analysis, Wiley Online Library, 2005.

[37] B. J. Frey, D. Dueck, Clustering by passing messages between data points, Science 315 (5814) (2007) 972–976.

[38] P. K. Agarwal, S. Har-Peled, N. H. Mustafa, Y. Wang, Near-linear time approximation algorithms for curve simplification, Algorithmica 42 (3-4) (2005) 203–219.

[39] C.-C. Chang, C.-J. Lin, LIBSVM: a library for support vector machines, ACM Transactions on Intelligent Systems and Technology 2 (3) (2011) 27:1–27:27.

[40] Y. Tang, Y.-Q. Zhang, N. V. Chawla, S. Krasser, SVMs mdeling for highly imbalanced classification, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 39 (1) (2009) 281–288.

[41] T. Fawcett, An introduction to roc analysis, Pattern Recognition Letters 27 (8) (2006) 861–874.

[42] A. Agathos, I. Pratikakis, S. Perantonis, N. S. Sapidis, Protrusion-oriented 3D mesh segmentation, The Visual Computer 26 (1) (2010) 63–81.

[43] FlowString: Partial streamline matching, http://www3.nd.edu/~cwang11/flowstring.html, accessed: 2015-3-28.

[44] J. Tao, C. Wang, C.-K. Shene, R. A. Shaw, A vocabulary approach to partial streamline matching and exploratory flow visualization, IEEE Transactions on Visualization and Computer GraphicsAccepted.