2015

EXTRACTING FLOW FEATURES USING BAG-OF-FEATURES AND SUPERVISED LEARNING TECHNIQUES

Yifei Li
Michigan Technological University, yifli@mtu.edu

EXTRACTING FLOW FEATURES USING BAG-OF-FEATURES AND SUPERVISED LEARNING TECHNIQUES

By

Yifei Li

A DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

In Computer Science

MICHIGAN TECHNOLOGICAL UNIVERSITY

2015

This dissertation has been approved in partial fulfillment of the requirements for the Degree of DOCTOR OF PHILOSOPHY in Computer Science.

Department of Computer Science

Dissertation Co-advisor: *Ching-Kuang Shene*

Dissertation Co-advisor: *Chaoli Wang*

Committee Member: *Song-Lin Yang*

Committee Member: *Scott A. Kuhl*

Department Chair: *Min Song*

Contents

List of Figures

List of Tables

Preface

This dissertation is original, published, independent work by the author, Yifei Li.

Acknowledgments

First of all, I would like to thank my advisor Dr. Shene and my co-advisor Dr. Wang for their guidance. Dr. Shene gave me tons of useful tips for writing a good paper in English, and Dr. Wang introduced me to the field of flow visualization.

Second, I would like to thank graduate school for giving me extensions to finish my degree.

Third, I would like to thank the Computer Science department at Michigan Tech for providing me financial support during my seven years PhD study. In my last year at Michigan Tech, the department also helped me find graduate assistance-ship from IT department. This financial support was very important to me and my family.

Finally, I want to thank my family for their continuous support and understanding during my PhD study. I would not be able to finish my degree without them.

Abstract

Measuring the similarity between two streamlines is fundamental to many important flow data analysis and visualization tasks such as feature detection, pattern querying and streamline clustering. This dissertation presents a novel streamline similarity measure inspired by the bag-of-features concept from computer vision. Different from other streamline similarity measures, the proposed one considers both the distribution of and the distances among features along a streamline. The proposed measure is tested in two common tasks in vector field exploration: streamline similarity query and streamline clustering. Compared with a recent streamline similarity measure, the proposed measure allows users to see the interesting features more clearly in a complicated vector field.

In addition to focusing on similar streamlines through streamline similarity query or clustering, users sometimes want to group and see similar features from different streamlines. For example, it is useful to find all the spirals contained in different streamlines and present them to users. To this end, this dissertation proposes to segment each streamline into different features. This problem has not been studied extensively in flow visualization. For instance, many flow feature extraction techniques segment streamline based on simple heuristics such as accumulative curvature

or arc length, and, as a result, the segments they found usually do not directly correspond to complete flow features. This dissertation proposes a machine learning-based streamline segmentation algorithm to segment each streamline into distinct features. It is shown that the proposed method can locate interesting features (e.g., a spiral in a streamline) more accurately than some other flow feature extraction methods. Since streamlines are space curves, the proposed method also serves as a general curve segmentation method and may be applied in other fields such as computer vision.

Besides flow visualization, a pedagogical visualization tool DTEvisual for teaching access control is also discussed in this dissertation. Domain Type Enforcement (DTE) is a powerful abstraction for teaching students about modern models of access control in operating systems. With DTEvisual, students have an environment for visualizing a DTE-based policy using graphs, visually modifying the policy, and animating the common DTE queries in real time. A user study of DTEvisual suggests that the tool is helpful for students to understand DTE.

Chapter 1

Introduction

Most fluids (e.g., air or water) are transparent, and their flow patterns are invisible. Flow visualization is the art of making flow patterns visible. It has been a central topic in scientific visualization for more than two decades. A flow represents the movement of a set of points in a fluid over time. Given the trajectory of any point in the flow, the velocity of the point at any time is simply a tangent vector to the trajectory. The velocity vectors of all the points in the flow at a given time consist of a vector (velocity) field. Conversely, given a velocity field where the velocity vectors change continuously, a flow can also be constructed (Section 1.1). If a flow induces a vector field or if a vector field produces a flow, people often use the words "vector field" and "flow field" interchangeably, and use vector (*resp.*, flow) field to emphasize the nature and properties of the vectors (*resp.*, flow). Vector fields are commonly seen

in many scientific, engineering and medical disciplines. For example, Figure 1.1 shows a vector field representing the air flow around a sports car, where the arrows indicate the direction of flow. The speed of flow is color coded with red being the fastest and blue being the slowest. Visualizing the air flow can help automobile designers spot potential problems early in the design process.

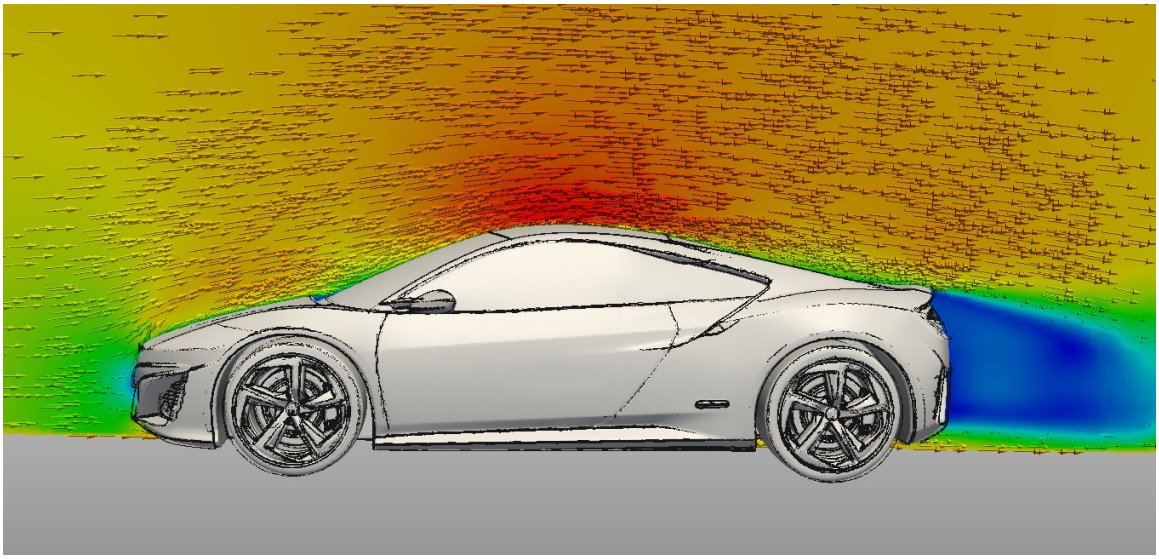
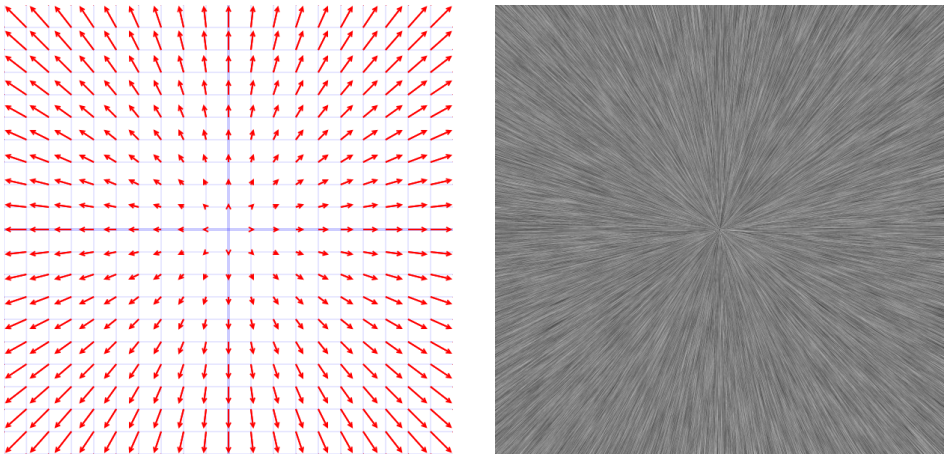
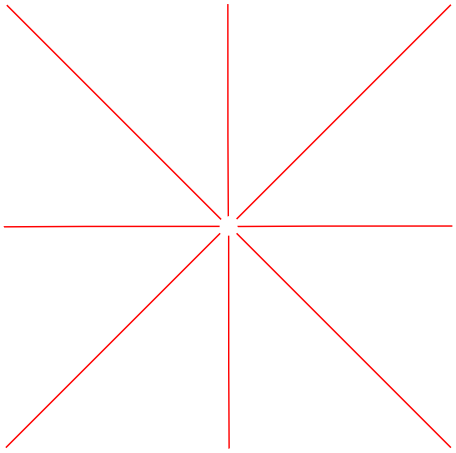


Figure 1.1: Aerodynamics analysis of a sports car. Image from *simscale.com*

The challenges for flow visualization include effectively visualizing both magnitudes and directions of vector data. In the past, various flow visualization techniques have been developed, which can be broadly categorized into glyph-based [66], texture-based [45] and geometric-based [28, 58] approaches. Figure 1.2 illustrates how these methods help to visualize a 2D vector field. Glyph-based approaches (Figure 1.2 (a)) simply renders an arrow for each vector to indicate the vector's direction, and the



(a) Glphy-based (b) Texture-based



(c) Geometric-based

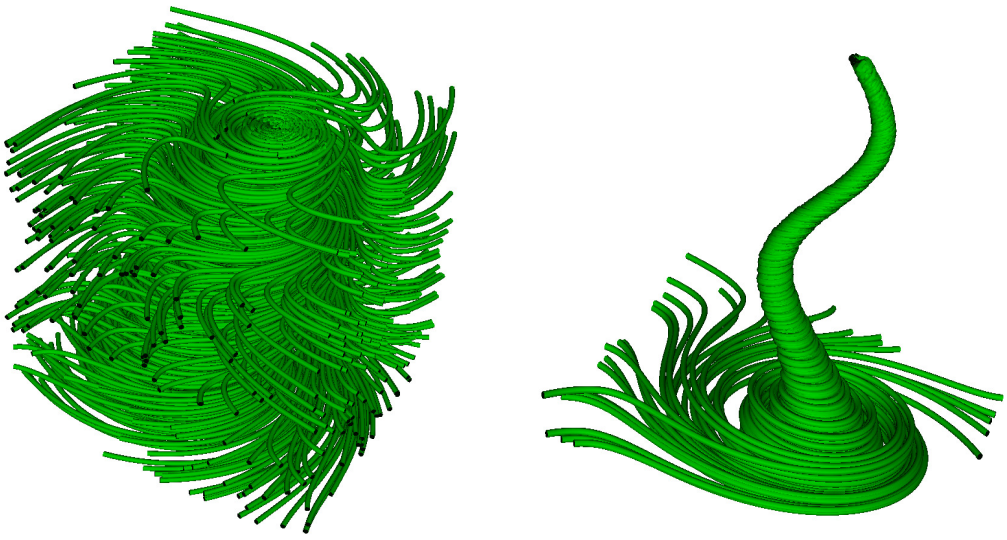
Figure 1.2: Different flow visualization techniques.

size of each arrow indicates the magnitude of the associated vector. Texture-based approaches (Figure 1.2 (b)) compute a texture which provides a detailed view of a vector field. Geometric-based approaches (Figure 1.2 (c)) use *streamlines* to depict a vector field. A streamline is a curve which is tangent to the vector at every point it passes in a vector field. In order to trace a streamline, imagine that a moving particle is placed in a vector field. The trajectory of the moving particle can be determined using the Runge-Kutta fourth-order method [35], which is a numerical method of

solving ordinary differential equations. This dissertation uses streamlines to visualize vector fields due to its popularity.

Recently, the research on flow visualization has focused on the problem of extracting flow features from a vector field. This problem becomes especially important when hundreds or thousands of streamlines are used to depict a vector field. The reason is that a large number of streamlines usually lead to visual clutter and occlusion, which makes it impossible for users to see interesting flow features. For example, in Figure 1.3 (a), a vector field representing a tornado event is visualized using 500 streamlines, which already looks cluttered. Figure 1.3 (b) shows the long spirals in the center of the tornado, which are occluded by the surrounding streamlines. Many techniques [68] based on physical or mathematical properties of flows have been proposed to extract flow features in a vector field. However, as suggested by [68], these methods may not work well for complex vector fields because the mathematical formula for detecting features like vortices (i.e., flows rotating around some axis) may not always hold.

Developing computer algorithms to automatically extract interesting features is a challenging problem because there is no rigorous definition of features. Moreover, different applications may look for different interesting features. For example, finding vortices is important in climate modeling because they may indicate the presence of a tornado. Figure 1.3 (b) illustrates an example of vortices. In aircraft design,



(a) A vector field visualized using 500 streamlines

(b) The streamlines in the center of the vector field are invisible in the left figure

Figure 1.3: The problem of visual clutter and occlusion.

shock waves (i.e., waves moving faster than sound) are important phenomena to study because they can cause structural failure in aircraft. Figure 1.4 shows an example of shock waves. These interesting features usually can be easily recognized in flow visualization by human but not by computers. Many approaches have been successfully developed in computer vision to define and extract features. This inspires the author to apply those techniques to the problem of flow feature extraction.

This dissertation attempts to solve the problem of flow feature extraction by leveraging the techniques used in computer vision and pattern recognition. In computer vision, users can specify what features they are interested in either by carefully deciding the ingredients in a feature, or simply providing computers with some example

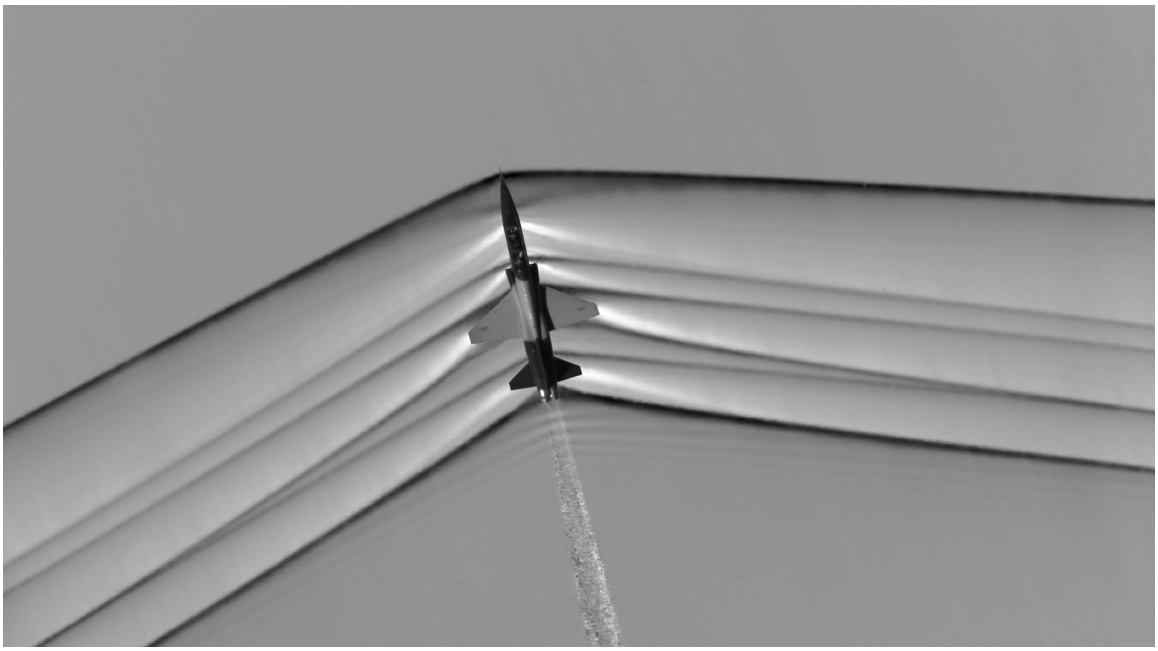


Figure 1.4: The shock wave of a supersonic jet flying over the Mojave Desert. Image from the NASA website.

features. Both approaches are applied in this dissertation. The results are encouraging, and it is worthwhile to experiment other computer vision related techniques in the future for flow visualization.

In the remaining of this chapter, Section 1.1 introduces some basic terms in flow visualization, Section 1.2 briefly discusses the motivation behind the work presented later in this dissertation, Section 1.3 summarizes the contributions made by this dissertation, and finally Section 1.4 gives an overview of the remaining chapters.

1.1 Terminology

This section defines the important concepts used throughout this dissertation. Section 1.1.1 explains vector fields and how they are specified for flow visualization applications. Section 1.1.2 gives the definition of a flow. Section 1.1.3 discusses the relationship between vector fields and flows, and when vector fields can be considered as flow fields. Section 1.1.4 defines steady and unsteady flows. Section 1.1.5 explains what streamlines are and how they are generated. Finally, Section 1.1.6 gives the definitions of different types of critical points.

1.1.1 Vector Fields

A function is of class C^k (or C^k continuous), where $k > 0$, if the derivatives $f', f'', \ldots, f^{(k)}$ exist and continuous. A C^0 function is a continuous function. A C^k *vector field* on $U \subseteq \mathbb{E}^n$ is a C^k mapping $V : U \to \mathbb{E}^n$ from an open set $U \subseteq \mathbb{E}^n$. Intuitively, a vector field assigns an n-dimensional vector to each point in a region of the n-dimensional Euclidean space.

In flow visualization, the open set U on which a vector field is defined is usually given as a 2D or 3D grid which consists of unit squares or cubes. The vector field assigns

vectors to the vertices of the grid. Figure 1.5 shows an example of a vector field which is represented as a 4×5 grid. The blue arrows at the grid vertices indicate velocity vectors.

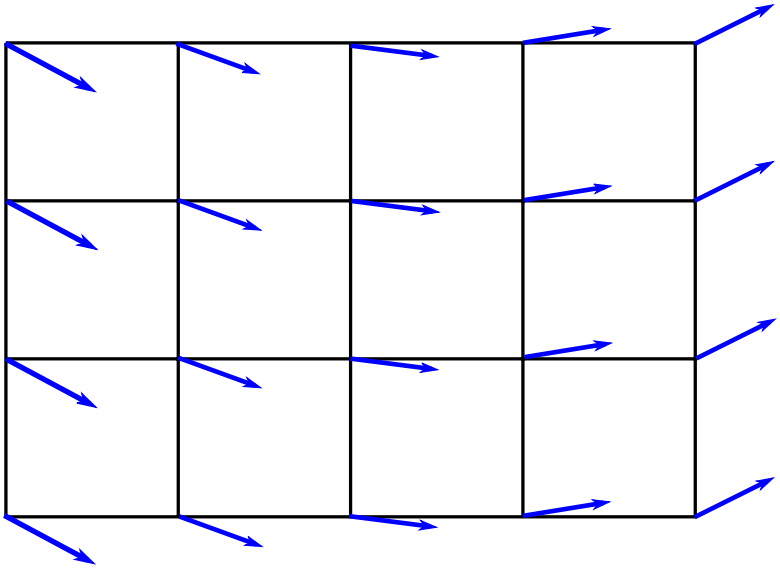


Figure 1.5: A 4×5 grid representing a 2D vector field: the velocity vectors (in blue) are specified at grid vertices.

1.1.2 Flow

A *flow* represents the movement of a set of points in a fluid over time. Formally, a flow on an open set U is a mapping $\phi : U \times \mathbb{R} \to U$ which satisfies the following two equations:

$$\begin{aligned} \phi(x, 0) &= x \\ \phi(\phi(x, t), s) &= \phi(x, s + t) \end{aligned}$$

where $x \in U$ and $s, t \in \mathbb{R}$.

For each point $x \in U$, its position at time t is $\phi(x, t)$ and its "initial" location is $\phi(x, 0) = x$. The set of points $\{\phi(x, t) : t \in \mathbb{R}\}$ is referred to as the *orbit* of $x \in U$ under the map ϕ. This set of points describes the trajectory of the movement of x over time. The second equation $\phi(\phi(x, t), s) = \phi(x, s+t)$ indicates that a point $x \in U$ whose location is $\phi(x, t)$ at time t moves to $\phi(\phi(x, t), s)$ after an additional time s, and the new location is $\phi(x, t + s)$. A flow ϕ is fully determined by the union of all its orbits.

1.1.3 Flow Fields

In flow visualization, people often use the words "vector fields" and "flow fields" interchangeably. In order to do so, it is required that either a flow can induce a vector field or a vector field can produce a flow. The remaining of this section explains the conditions that need to be met for a flow to induce a vector field and for a vector field to produce a flow.

A C^1 flow $\phi : U \times \mathbb{R} \to U$ defined on an open set U induces a C^0 vector field. The trajectory of any point x in this flow can be represented by the function $\phi_x(t) = \phi(x, t)$ with the time variable t. Since ϕ is C^1, the derivative of $\frac{d(\phi_x(t))}{dt}\Big|_{t=0}$ exists and is continuous, which is the velocity vector at point $\phi_x(0)$ for time $t = 0$. Therefore, a

vector field $V(x) = \dot{\phi_x}(0)$ is obtained, which assigns the velocity vector at time $t = 0$ to every point $x \in U$. This vector field V is C^0 (*i.e.*, continuous) because the flow ϕ is C^1. It is also referred to as the *velocity field* of the flow ϕ at time $t = 0$. Note that if the flow ϕ is not C^1, it may not be possible to construct a vector field from it.

Conversely, a C^0 vector field can produce a flow ϕ so that the given vector field is the induced velocity field of ϕ. As mentioned earlier, a flow is the union of the orbit of every point in the flow. Given the velocity field $V : U \to \mathbb{E}^n$ of the unknown flow and an arbitrary point x in the flow, the orbit of x is a curve $C : \mathbb{R} \to U$ with an initial point $C(0) = x$. The tangent vector at $C(t)$ is given by $V(C(t))$. Therefore, to find the orbit $C(t)$ of point x, the following first order ordinary differential equation needs to be solved:

$$\begin{aligned} \dot{C}(t) &= V(C(t)) \\ C(0) &= x \end{aligned} \tag{1.1}$$

Since the vector field V is C^0, the above equation has a unique solution $C(t)$ according to the existence and uniqueness of ordinary differential equation. The corresponding flow can be defined as $\phi(x, t) = C_x(t)$ after the orbit for every point x is obtained. This flow is C^1 because it has continuous derivatives along each orbit as guaranteed by the C^0 velocity field V. Note that if the velocity field V is not C^0, it may not be possible to construct a flow so that V is the induced velocity field of the flow.

Based on the above discussion, the words "vector field" and "flow field" can be used

interchangeably if a C^1 flow on an open set induces a C^0 vector field on that set, or a C^0 vector field produces a C^1 flow so that the vector field is the velocity field of the flow. People use vector (*resp.*, flow) field to emphasize the nature and properties of the vectors (*resp.*, flow).

1.1.4 Steady and Unsteady Flows

A *steady* flow is a flow in which the properties assigned to any point are independent of the time parameter. Because this dissertation only focuses on the velocity at each point, a flow is steady if the vector assigned to any point does not vary over t. A flow that is not steady is an *unsteady* flow.

1.1.5 Streamlines

Given a point $x \in U$ in a flow $\phi : U \times \mathbb{R} \to U$, its orbit $\phi_x(t)$ is also called a *streamline* in flow visualization. As mentioned above, a streamline with an initial position $\phi_x(0) = x$ can be found by solving the ordinary differential equation (Equation (1.1)). Due to the uniqueness of the solution, no two streamlines in a flow can cross each other at any given time t.

1.1.6 Critical Points

A *critical point* in a vector field is a point whose associated vector is zero. A critical point may be a *source* (where vectors emanate from a point), *sink* (where vectors converge into a point), *saddle* (where vectors repel each other at a point), or *spiral* (where vectors revolve around a point).

1.2 Motivation

Measuring the similarity between two streamlines is a fundamental task in flow visualization. For example, streamline clustering [80, 94, 95] relies on similarity measures to group streamlines with similar shapes together. For vector fields visualized by a large number of streamlines, streamline clustering has the potential to alleviate visual clutter and occlusion because it allows users to focus on one group of similar streamlines at a time without being interfered by other streamlines.

Distribution-based streamline similarity measures are popular nowadays. They describe a streamline using some feature distribution (e.g., histograms of curvature), and measure the similarity between two streamlines as the distance between their distributions. Since a streamline's feature distribution is not affected by its distance

to other streamlines, distribution-based methods are independent of the relative positions of two streamlines. One problem with distribution-based similarity measures is that it does not consider the distances among different features along a streamline. This may cause two dissimilar streamlines to be considered as similar because of their similar distributions of features. This is illustrated in Figure 1.6. Although the two curves look differently, the curve in (a) is actually made from reshuffling different parts of the curve in (b). This dissertation addresses this issue by proposing a

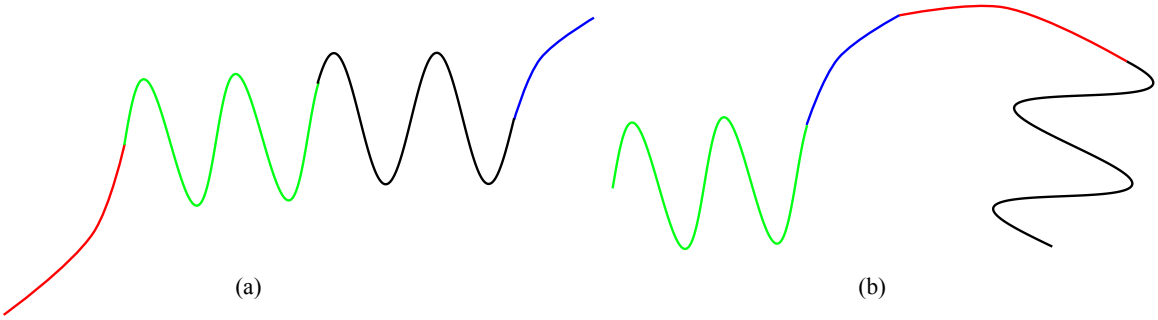
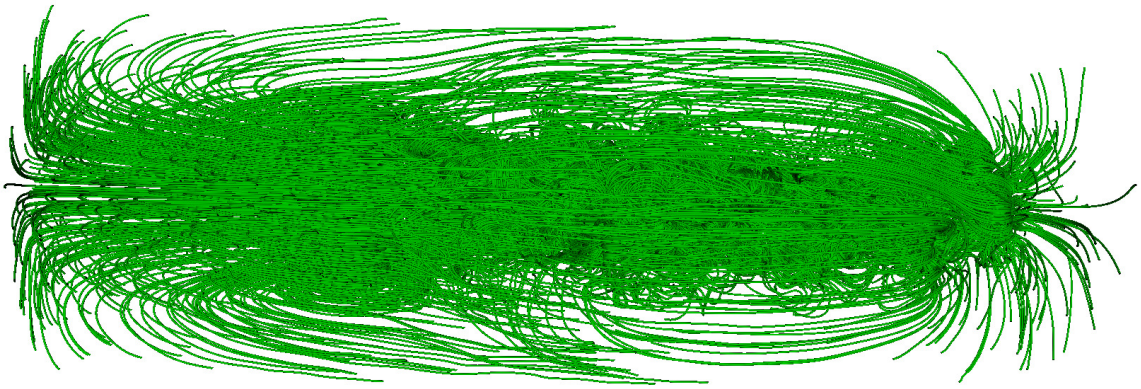


Figure 1.6: The order of features along a curve is important to similarity measures.

streamline descriptor which encodes the relative positions of different features along a streamline.

Sometimes streamline clustering still fails to show the detailed flow features to users. For example, users may be more interested in seeing the part of a streamline in the vicinity of a critical point, but that part may still be occluded after streamline clustering. Figure 1.7 (a) shows a vector field visualized with 600 streamlines, and Figure 1.7 (b) shows a cluster after the streamlines are clustered. In this cluster, some spirals (in red circle) still cannot be seen clearly. One way to overcome this issue is



(a) A vector field visualized with 600 streamlines

(b) After clustering, some sprials still cannot be seen clearly

Figure 1.7: Some features may still be occluded after clustering.

to segment every streamline such that different features on each can be separated. These different features can then be clustered based on shape similarity such that each type of feature can be viewed individually. This is a challenging task because it is very hard to clearly define what a feature is. In computer vision, this type of task is usually handled through machine learning, where users tell the system what features they are looking for in the hope that computers can find similar features in other scenarios. Machine learning is also used in this dissertation to solve the streamline segmentation problem.

1.3 Contributions

This dissertation studies two fundamental problems in flow visualization: streamline similarity measure and streamline segmentation. Effective flow visualization relies on the solutions to these two problems in order to detect and query interesting flow features. Since flow visualization is widely used in many engineering disciplines for analyzing complex vector fields, the contributions made by this dissertation may be applied to help solve various engineering problems. For example, in order to assess the risk of aneurysm rupture [6, 20], patient-specific hemodynamic data is usually visualized with a dense set of streamlines [63]. A clustering algorithm can use the proposed streamline similarity measure to cluster the blood flow into different patterns, which will help the diagnostics. Another application is in ocean prediction [34], where detecting flow features such as vortices is essential for effective prediction of the ocean environment. Due to the vastness of ocean, it is important to concentrate measurements in the regions where one can observe important physical features. The proposed streamline segmentation technique may be applied to find vortices in the visualization of ocean flow.

Streamline similarity measure and streamline segmentation are two important tasks in flow feature extraction. Feature extraction answers the question of what should be visualized in a vector field. A good flow feature extraction method requires solving

a few subtasks. First of all, features should be properly described before they can be extracted. After extraction, how to present the extracted features is another problem. One possible way is to cluster the extracted features based on similarity. This in turn introduces the problem of streamline similarity measure because most clustering algorithms require calculating the similarity between two input objects.

The contribution of this dissertation lies in the design of a few novel solutions to the above problems. These solutions are inspired by some widely used ideas from computer vision and pattern recognition. This dissertation makes some early attempts to apply those ideas to flow visualization. In summary, the contributions made by this dissertation include:

- Describing a streamline (or any part of it) in such a way that not only the distribution of features on the streamline but also their relative order along the streamline are considered. The similarity between two streamlines can be computed as the distance between their corresponding descriptors. Unlike the distanced-based similarity measures, the proposed streamline similarity measure is independent of the relative positions and orientations of streamlines. Compared with other distribution-based methods, it encodes the relative positions of different features in a streamline's feature description, and does not require any step to partition a streamline first (Section 2.1). By leveraging GPUs, it can be computed much faster than existing methods. These properties make it

able to more accurately filter out in real time flow features not interesting to users.

- Leveraging machine learning to segment a streamline so that distinct features on it can be separated. The proposed method gives users a fine level of control on the type of features that can be extracted. Therefore, it can further reduces visual clutter and occlusion compared with streamline clustering. This is the early attempt of applying machine learning to flow feature extraction. The successful application may encourage more future research on how to use machine learning to assist flow visualization. The proposed method may also be applied to the curve segmentation problem in computer vision. In computer vision, the segmentation of the silhouette of a 2D shape can be used to describe the features of the 2D shape [93].

- Proposing a new heuristic which can help determine whether to separate two features or not in the above segmentation process. Experiments showed that this heuristic improves the final segmentation results. This heuristic may also be combined with other properties to improve similarity measures between two streamlines.

1.4 Dissertation Organization

In the following, Chapter 3 introduces a streamline similarity measure leveraging bag-of-features technique. Chapter 4 provides a detailed description of a flow feature extraction method based on supervised machine learning. Chapter 5 presents a minor research focus, which is developing a pedagogical visualization tool for teaching access control in operating systems.

Chapter 2

Related Work

This chapter reviews the related work on streamline similarity measure and streamline segmentation. The following is the organization of this chapter. Section 2.1 reviews two major types of streamline similarity measure: distanced-based and distribution-based measures, and then compares the proposed similarity measure with some existing ones. Section 2.2 discusses a few existing streamline segmentation algorithms which are fundamental to feature extraction, and points out the advantages of the supervised streamline segmentation algorithm.

2.1 Streamline Similarity Measures

Distanced-based Similarity Measures. These streamline similarity measures are defined in terms of the distance between two streamlines X_i and X_j, which are given as two polygonal curves with N vertices $\mathbf{p}_{k(1\leq k\leq N)}$ and M vertices $\mathbf{p}_{l(1\leq l\leq M)}$ each. For example, the *closest point distance* [59] d_c is the closest distance between any two points from X_i and X_j:

$$d_c(X_i, X_j) = \min_{\mathbf{p}_k \in X_i,\ \mathbf{p}_l \in X_j} \|\mathbf{p}_k - \mathbf{p}_l\|$$

where $\|.\|$ is the Euclidean distance.

The *mean of closest point distances* [25] d_μ is the average of $\tilde{d}_\mu(X_i, X_j)$ and $\tilde{d}_\mu(X_j, X_i)$, where $\tilde{d}_\mu(X_i, X_j)$ (resp., $\tilde{d}_\mu(X_j, X_i)$) is the mean of the distances between each point $\mathbf{p}_k$ (resp., $\mathbf{p}_l$) on X_i (resp., X_j) to its closest point on X_j (resp., X_i):

$$d_\mu(X_i, X_j) = \frac{1}{2}\left(\tilde{d}_\mu(X_i, X_j) + \tilde{d}_\mu(X_j, X_i)\right),$$
$$\text{where } \tilde{d}_\mu(X_i, X_j) = \frac{1}{N}\sum_{\mathbf{p}_k \in X_i}\left(\min_{\mathbf{p}_l \in X_j} \|\mathbf{p}_k - \mathbf{p}_l\|\right)$$

The *Hausdorff distance* [59] d_H is the maximum between $\tilde{d}_H(X_i, X_j)$ and $\tilde{d}_H(X_j, X_i)$, where $\tilde{d}_H(X_i, X_j)$ (resp., $\tilde{d}_H(X_j, X_i)$) is the maximum of all the distances between $\mathbf{p}_k$

(resp., $\mathbf{p}_l$) on X_i (resp., X_j) to its closest point on X_j (resp., X_i):

$$d_H(X_i, X_j) = \max(\tilde{d}_H(X_i, X_j), \tilde{d}_H(X_j, X_i)),$$
$$\text{where } \tilde{d}_H(X_i, X_j) = \max_{\mathbf{p}_k \in X_i} \left(\min_{\mathbf{p}_l \in X_j} \|\mathbf{p}_k - \mathbf{p}_l\| \right)$$

Distribution-based Similarity Measures. The most significant issue of distance-based similarity measures is that similar streamlines far away from each other will not be considered as similar. In order to overcome this problem, similarity measures proposed recently usually view each streamline as a distribution of features. The similarity between two streamlines is computed as the distance between their corresponding distributions. For instance, McLoughlin et al. [57] computed for each streamline a histogram where each bin of the histogram is the sum of curvature, torsion and tortuosity values of all the points falling into that bin. Then they performed similarity comparisons using χ^2 on those histograms. Lu et al. [54] proposed to take into account the order of features along a streamline in addition to the distribution of features. They first segmented each streamline and computed a histogram of curvature, torsion and curl for each segment. Each streamline is then represented by a set of histograms. The distance between two streamlines is computed by minimizing the distance between two sets of histograms. More specifically, assume that two streamlines X and Y have N and M segments, respectively. These two streamlines can be represented by their corresponding sets of histograms $H_X = (x_1, x_2, \ldots, x_N)$

and $H_Y = (y_1, y_2, \ldots, y_M)$, where x_i (resp., y_j) is a histogram for the i-th (resp., j-th) segment of streamline X (resp., Y). The similarity between streamlines X and Y is computed by finding a mapping V between H_X and H_Y which can minimize the sum of distances between each pair of histograms in the mapping. This mapping V is obtained using an algorithm called dynamic time warping (DTW). DTW [60] is a method which calculates an optimal match between two given sequences (e.g., sequences of histograms) such that the dissimilarity between the two sequences can be minimized. Suppose the mapping found by DTW is $V = (v_1, v_2, \ldots, v_L)$, where $v_{l(1 \leq l \leq L)} = (n_l, m_l) \in [1, N] \times [1, M]$. The mapping V has L pairs of histograms between H_X and H_Y. The distance between H_X and H_Y is calculated as follows:

$$\text{Dist}(H_X, H_Y) = \min_V \left(\sum_{l=0}^{L} d(x_{n_l}, y_{m_l}) \right) \tag{2.1}$$

where $d(x_{n_l}, y_{m_l})$ is the L_1-distance between two histograms.

Proposed Solution. This dissertation proposes a novel distribution-based similarity measure which addresses the shortcomings of the method by Lu et al. [54]. First of all, their method requires segmentation as a preprocessing step, and the results of the segmentation will affect the final similarity measure. The purpose of this step is to make sure the order of features along a streamline is considered in the final similarity measure. This requires users to set two parameters to determine when the segmentation should stop: either when the current segment is too short or cannot

be separated into two dissimilar enough segments. Both of these two parameters are not intuitive for users. Second, as shown by Equation (2.1), their method requires a way of measuring the similarity between two segments before obtaining the similarity measure between two streamlines.

To overcome these shortcomings, this dissertation proposes to describe a streamline in such a way that not only the distribution of features but also the distance between different features along a streamline are considered. In other words, the distance among features is automatically encoded in the descriptor for a streamline. The similarity between two streamlines is measured through the L_1-distance between their corresponding descriptors (i.e., vectors). Compared with the similarity measure by Lu et al. [54], it is less computationally expensive because (1) no segmentation is required, and (2) the similarity measure is a straightforward computation of the distance between vectors. Comparison results also show that the proposed similarity measure performs better in streamline query and clustering applications.

2.2 Streamline Segmentation

Streamline segmentation is often used as an important step in flow feature extraction. For instance, Section 2.1 already mentions that Lu et al. [54] performed streamline segmentation such that a 1D histogram can be computed for each segment and then

used during streamline similarity comparison. Wang et al. [89] partitioned a streamline into the 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 direction. The segments in the final segmentation are considered as flow features. Both of these methods locate segmentation points by checking whether the two sides of a point correspond to different features. They use multiple criteria to measure the similarity between the two sides of a potential segmentation point. However, the problem of how to combine different criteria is a challenging task. Different criteria may not play equally important roles in determining the segmentation, and the importance of each criterion may change depending on the scenarios.

Tao et al. [84] segmented a streamline such that the accumulated curvature of each segment does not exceed a certain threshold. A number of short segments will be generated, and each one of them is assigned a character. Frequent text patterns are extracted and treated as flow features. One disadvantage of this method is that the final extracted features depend on two parameters used for finding frequent text patterns in a collection of strings.

The problem of curve segmentation has also been studied in the computer vision community. Computer vision applications usually focus on how to segment a curve into a combination of representations such as lines, elliptical and superelliptical arcs [70]. In flow visualization, since users are usually interested in flow features corresponding

to more complicated curves such as spirals, the curve segmentation techniques used in computer vision are not suitable.

Proposed Solution. This dissertation proposes to use machine learning to solve the problems of previous streamline segmentation methods. The proposed method also uses multiple criteria to determine the similarity between the two sides of a potential segmentation point. Instead of requiring users to adjust the values of different criteria individually for segmentation, the proposed method constructs a non-trivial decision function of these criteria which determines whether a point is a segmentation point or not. The resulting segments correspond to desired features, and no further filtering is required. Another advantage of using machine learning is that it gives users the flexibility of segmenting similar streamlines differently for different applications.

The proposed streamline segmentation works as follows: (1) users will be asked to specify what types of features they are interested in by segmenting a few example streamlines; and (2) machine learning is leveraged to segment other streamlines so that user-desired features can be extracted. Comparison results show that the proposed method outperforms several other methods, which encourages future research on applying machine learning to flow feature extraction.

Chapter 3

Streamline Similarity Analysis using Bag-of-Features

3.1 Overview

Visual clutter is a major issue when a large number of streamlines are rendered to depict a vector field. Due to visual clutter, it is difficult for users to explore the underlying features. Clustering the streamlines based on their shape similarities is an effective way to address this problem. However, designing a metric to evaluate streamline similarity is a challenging task. Inspired by the idea of *bag-of-features* widely used in computer vision, the following method of evaluating the similarity

between two streamlines is presented. First, *feature descriptors* are computed at each point along a streamline for all the streamlines. Second, *quantization* is carried out to obtain a compact representation of the *descriptor space* which consists of all the feature descriptors computed in the first step. Third, *spatially sensitive bag-of-features* is constructed for each streamline based on the quantization results from the second step. Finally, the similarity between two streamlines is calculated as the *weighted Manhattan distance* between their corresponding spatially sensitive bag-of-features. Compared with previous streamline similarity metrics, this metric (1) is invariant to rotation and translation, and (2) takes into consideration the spatial relationship among different feature descriptors. In practice, this metric makes it possible to find out the streamlines which are more visually similar to a query streamline. The utility of this approach is demonstrated by two common tasks in flow field exploration: streamline similarity query and streamline clustering. This work has been published in *IS&T/SPIE Conference on Visualization and Data Analysis 2014* [48].

3.2 Background

This section first introduces Shannon entropy (Section 3.2.1), and then explains curvature and torsion (Section 3.2.2). These important concepts will also be used in the next chapter. At last, the concept behind bag-of-features model (Section 3.2.3) is discussed.

3.2.1 Shannon Entropy

Entropy [26] is a measure of the uncertainty of a random variable. Let X be a discrete random variable with alphabet χ and probability mass function $p(x) = \Pr\{X = x\}, x \in \chi$. The entropy $H(X)$ of a discrete random variable X is defined by

$$H(X) = -\sum_{x \in \chi} p(x) \log p(x) \tag{3.1}$$

The log is to the base 2 and entropy is expressed in bits. For example, the entropy of a fair coin toss is 1 bit. The convention $0 \log 0 = 0$ is used, which is easily justified by continuity since $x \log x \rightarrow 0$ as $x \rightarrow 0$. If the base of the logarithm is e, the entropy is measured in *natural unit of information* (nat). This dissertation uses base 2 logarithms only, and hence all the entropies will be measured in bits.

The entropy of X can also be interpreted as the expected value of the random variable $\log \frac{1}{p(x)}$. Therefore, it can also be written as:

$$H(X) = \sum_{x \in \chi} p(x) \log \frac{1}{p(x)} \tag{3.2}$$

Since $0 \leq p(x) \leq 1$ implies that $\log \frac{1}{p(x)} \geq 0$, entropy $H(X)$ is always non-negative.

3.2.2 Curvature and Torsion

Intuitively, *curvature* [69] is a measure of how 'curved' a curve is. Let $\gamma(t) : \mathbb{R} \to \mathbb{R}^3$ be a *regular* (i.e., everywhere differentiable) curve in $\mathbb{R}^3$. Then, its curvature is

$$\kappa = \frac{\|\dot{\gamma} \times \ddot{\gamma}\|}{\|\dot{\gamma}\|^3} \tag{3.3}$$

where $\dot{\gamma}$ and $\ddot{\gamma}$ are the first and second derivatives. The *center of curvature* at a point $\gamma(t)$ is the point q such that a circle centered at q which meets the curve at $\gamma(t)$ have the same tangent and curvature as the curve has there.

While a plane curve is essentially determined by its curvature, this is no longer true for space curves. For example, a circle of radius of one in the xy-plane and a circular helix $\gamma(\theta) = (\frac{1}{2}\cos\theta, \frac{1}{2}\sin\theta, \frac{1}{2}\theta)$ both have curvature one everywhere, but it is obviously impossible to change one curve into another by any combinations of rotations or translations. Only rotations and translations are performed because curvature is invariant to these two types of transformations, and the two curves have the same curvature everywhere. *Torsion* is introduced to measure how sharply a curve is twisting out of the *plane of curvature*. The plane of curvature at a point on a curve is a plane determined by the tangent and the center of curvature at that point. The

torsion for a regular curve $\gamma(t)$ is given by

$$\tau = \frac{(\dot{\gamma} \times \ddot{\gamma}) \cdot \dddot{\gamma}}{\|\dot{\gamma} \times \ddot{\gamma}\|^2} \tag{3.4}$$

where $\dddot{\gamma}$ is the third-order derivative.

It can be proved that the curvature and torsion of a curve together determine the curve up to a rigid motion [69].

3.2.3 Bag-of-Features

The bag-of-features method is widely used in various computer vision tasks such as image classification and object detection [64]. The name comes from the bag-of-words representation used in textual information retrieval. With bag-of-words, one represents a document as a normalized histogram of word counts. Commonly, one counts all the words from a dictionary that appear in the document. This dictionary may exclude certain non-informative words such as articles (like "the"), and it may have a single term to represent a set of synonyms. The term vector that represents the document is a sparse vector where each element is a term in the dictionary and the value of that element is the number of times the term appears in the document divided by the total number of dictionary words in the document (and thus, it is also a normalized histogram over the terms). The term vector is the bag-of-words

document representation – called a "bag" because all ordering of the words in the document have been lost.

The bag-of-features image representation is analogous. More specifically, the following steps are performed to obtain such a representation for an image:

- **Feature point detection.** The main goal of feature point detection is to find points or regions in an image that carry significant information. There are different approaches to detect feature points/regions in an image. For example, scale-invariant feature transform (SIFT) [51] is an algorithm in computer vision to detect and describe local features in images. SIFT detects feature points by looking for local maxima of the discrete Laplacian at different scales. Another example is Maximum Stable Extremal Region (MSER) [43], which is a computer vision algorithm for detecting regions in an image that differ in intensity compared to surrounding regions. More specifically, it finds intensity level sets in the image which exhibit the smallest variation of area when traversing the level-set graph. Finally, it is also possible to select all the points in the image as the set of feature points.

- **Feature descriptor computation.** Each feature point needs to be described by a *feature descriptor*. A feature descriptor is simply a vector which contains the local image information in the neighborhood of the feature point. For example, SIFT computes a 128-dimensional feature descriptor constructed as

local histograms of image gradient orientations around a feature point. After this step, the feature points detected in the previous step can be represented by their corresponding feature descriptors.

- **Feature descriptor quantization.** Since the number of feature points is usually large, to reduce the representation size for an image, only a set of representative feature descriptors is used to represent an image. To achieve this, vector quantization is carried out in the descriptor space which consists of all the feature descriptors obtained in the previous step. The result of vector quantization is called a *vocabulary*, which is a set of representative feature descriptors in the descriptor space. These representative feature descriptors are called *'words'*. More specifically, the set of n feature descriptors $\{f_1, f_2, \ldots, f_n\}$ can be replaced by a vocabulary P which only contains V representative feature descriptors $\{p_1, p_2, \ldots, p_V\}$, where n is equal to the number of detected feature points and $V \leq n$. After quantization, each feature descriptor f_i can be replaced by the word p_j from the vocabulary such that the distance between f_i and p_j is less than that between f_i and p_k for all $k \neq j$. One commonly used distance measure is the Euclidean distance.

- **Bag-of-features construction.** Bag-of-features representation can be constructed as follows: for each word, count the number of feature descriptors which are represented by this word. In other words, a histogram which counts the frequency of appearance of each word is constructed. In this histogram,

since each bin corresponds to a word, the number of bins in this histogram is equal to the size of the vocabulary.

3.3 Spatially Sensitive Bag-of-Features

Since streamlines are usually sparse in features compared with images, every point on a streamline is considered as a feature point. The feature descriptor for each point is a vector consisting of all the features described in Section 3.4. Given all the feature descriptors, the first step is to quantize the descriptor space in order to obtain a vocabulary. A vocabulary $P = \{p_1, p_2, \cdots, p_V\}$ of size V is a set of representative vectors in the descriptor space. The dimension of each vector is the number of features computed at each point on a streamline. For example, the dimension of each representative vector is two if only curvature and torsion are computed at each point. The quantization can be done using the efficient k-means algorithm by Kanungo et al. [41], and the size of the vocabulary can be set empirically. Note that increasing the size of vocabulary does not necessarily improve streamline similarity comparison results because there may be many duplicates in the vocabulary due to the small number of features (Section 3.4) used. Compared with 2D images, it is much more difficult to find a large number of independent features for streamlines. In the following experiment, the size of vocabulary is set to 16.

Given a vocabulary, a feature distribution for a streamline point x is the following $V \times 1$ vector:

$$\theta(x) = (\theta_1(x), \theta_2(x), \cdots, \theta_V(x))^T \tag{3.5}$$

where $\theta_i(x) = 1$ and i is the index of the word in the vocabulary which best describes the features at point x, and $\theta_j(x) = 0$ for $j \neq i$. To obtain the bag-of-features for a streamline X, the feature distributions is simply summed up over the entire streamline:

$$\text{BoF}(X) = \sum_{x \in X} \theta(x) \tag{3.6}$$

where x is a point on the streamline X. In other words, a streamline is now represented by $\text{BoF}(X)$ which is a histogram of words, and hence common techniques which evaluate histogram similarity can be used to measure the similarity between two streamlines.

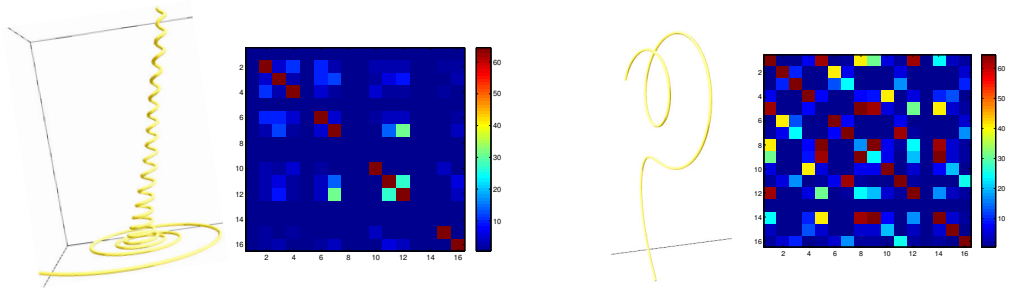
However, the above definition of bag-of-features only considers the distribution of words in the vocabulary and loses the spatial relationship among them. Take text search as an example, in a document about "matrix decomposition", the words "matrix" and "decomposition" are frequent. However, a document about the movie *Matrix* and a document about decomposition of organic matter will also contain these words. This will lead to similar word statistics and, consequently, similar bag-of-features. In order to overcome this issue, text search engines commonly use vocabularies not only of single words but also of combinations of words or *expressions*. For

instance, the expression "matrix decomposition" will be frequently found in a linear algebra textbook, but unlikely in a document about the movie *Matrix*.

In the case of streamlines, two visually dissimilar streamlines may have many similar words because of the small size of vocabulary. Therefore, the two streamlines may be considered similar because the distance between their bag-of-features is small. This issue is addressed by considering the contributions of all the possible pairs of words (i.e., expressions) from the vocabulary. As mentioned above, a feature descriptor is computed for each point on a streamline, and the feature descriptor can be represented by a certain word from the vocabulary. If two points are spatially close to each other on a streamline, the expression consisting of their words has a large contribution. More formally, the spatially sensitive bag-of-features for a streamline X is defined as follows:

$$\text{SS-BoF}(X) = \sum_{x \in X} \sum_{y \in X \wedge y \neq x} \theta(x)\theta^T(y) \frac{1}{d(x,y)/l} \tag{3.7}$$

where l is the total length of a streamline and $d(x, y)$ is the arc-length between two points x and y on that streamline. Notice that $\theta(x)$ is a $V \times 1$ column vector as explained above, so $\theta(x)\theta^T(y)$ is a $V \times V$ matrix. Since the vector only has one non-zero element equal to one, the elements of the matrix $\theta(x)\theta^T(y)$ also only has one non-zero element equal to one. The non-zero matrix element at row i and column j indicates an expression consisting of the i-th and j-th words from the vocabulary. The resulting spatially sensitive bag-of-features is a $V \times V$ symmetric matrix which represents the



(a) (b)

Figure 3.1: Two different streamlines along with the symmetric matrices representing their spatially sensitive bag-of-features.

contributions of different expressions. It is symmetric because $\theta(x)\theta^T(y) = \theta(y)\theta^T(x)$ and $d(x, y) = d(y, x)$. To make it easier to evaluate the distance between such two matrices, the symmetric matrix is turned into a $\frac{V(V+1)}{2}$ dimension vector by only considering its upper triangular part. Notice that $\frac{l}{d(x,y)}$ instead of just $d(x, y)$ is used as the denominator in order to offset scaling effects. Otherwise, a scaled version of a streamline will have a very different spatially sensitive bag-of-features than the original streamline since the values in the matrix will also be scaled. Figure 3.1 shows two streamlines along with the symmetric matrices representing their spatially sensitive bag-of-features. As it can be seen, the two symmetric matrices indeed look different for the two streamlines of dissimilar shapes.

When it comes to compare the bag-of-features representation of two streamlines, it is noticed that not all words are equally important for the purpose of comparison. In text retrieval, it is common to assign different weights to words according to their statistical importance. Down-weighting common words like prepositions and

articles increase the performance of text search engines. Similarly, down-weighting the contribution of the common expressions is also an effective technique which has been successfully used in object retrieval in video [82] and three-dimensional shape retrieval [16]. Similarly, *inverse document frequency* of expression i is defined as the logarithm of the inverse fraction of streamlines in a flow field in which this expression appears:

$$w_i = \log\left(\frac{D}{\sum_{j=1}^{D}\delta(f_i(X_j) > 0)}\right) \tag{3.8}$$

where D is the total number of streamlines, δ an indicator function, and $f_i(X_j)$ counts the number of occurrences of expression i in streamline j. The indicator function δ evaluates to one if $f_i(X_j) > 0$, otherwise to zero. The smaller w_i is, the more common the expression i is in all the streamlines, and so it is less likely to be able to discriminate between streamlines.

Finally, the weighted Manhattan distance is used to measure the similarity between two streamlines X and Y:

$$d(X,Y) = \sum_{i=1}^{V(V+1)/2} w_i|\text{SS-BoF}(X)_i - \text{SS-Bof}(Y)_i| \tag{3.9}$$

Again, because spatially sensitive bag-of-features is a symmetric matrix, only the upper triangular part of the matrix (a total of $\frac{V(V+1)}{2}$ matrix elements) are used to compute the distance. w_i is the weighting coefficient explained in the previous paragraph.

3.4 Streamline Feature Selection

Spatially sensitive bag-of-features makes it possible to consider a combination of suitable feature metrics. Since the goal is to find out streamlines with similar shapes, the first two metrics considered are curvature and torsion. According to the fundamental theory of curves [69], any regular curve has its shape completely determined by its curvature and torsion up to a rigid transformation. Section 3.4.1 gives details on how curvature and torsion are computed on streamlines. Curvature and torsion are examples of *local* geometric properties. Experiments show that similarity comparison results can be improved by considering the other two *global* geometric properties: velocity direction entropy (Section 3.4.2) and *tortuosity* (Section 3.4.3).

3.4.1 Computing Curvature and Torsion in Vector Fields

Instead of directly computing the curvature and torsion at a point on a streamline, they are first calculated at the grid vertices (Chapter 1) in a vector field, and then the tri-linear interpolation can be applied to compute them at other points in the vector field.

Equation 3.3 and Equation 3.4 are used to compute curvature and torsion at grid

vertices. These two equations require the second-order and third-order derivatives to be calculated first (first-order derivatives are velocities which are already known). Numerical methods such as central difference approximation can be used to approximate these derivatives. The central difference approximation can be derived as follows using Taylor approximation:

$$\begin{aligned}
\gamma(x+h) &= \gamma(x) + \dot{\gamma}(x)h + \frac{1}{2}\ddot{\gamma}(x)h^2 + \frac{1}{6}\dddot{\gamma}(x)h^3 \\
\gamma(x-h) &= \gamma(x) - \dot{\gamma}(x)h + \frac{1}{2}\ddot{\gamma}(x)h^2 - \frac{1}{6}\dddot{\gamma}(x)h^3 \\
\dot{\gamma}(x) &\approx \frac{\gamma(x+h) - \gamma(x-h)}{2h}
\end{aligned}$$

For a vector field, because the velocity at time t (i.e., $\gamma(t)$) is only available as input at the grid vertices and the grid are all unit squares/cubes, h is set to 1. Therefore, the second and third order derivatives at grid vertices can be approximated using the central difference method as follows:

$$\begin{aligned}
\ddot{\gamma}(x) &= \frac{\dot{\gamma}(x+1) - \dot{\gamma}(x-1)}{2} \\
\dddot{\gamma}(x) &= \frac{\ddot{\gamma}(x+1) - \ddot{\gamma}(x-1)}{2}
\end{aligned} \tag{3.10}$$

Once the curvature and torsion at each grid vertex is known, the tri-linear interpolation can be applied to compute the curvature and torsion at an arbitrary point (x, y, z) in a vector field. The tri-linear interpolation works by first locating the unit cube in the grid which contains the point (x, y, z). Then it applies the linear interpolation first along the x direction, then the y direction, and finally the z direction in

the unit cube.

Let (x_0, y_0, z_0) and (x_1, y_1, z_1) be the two diagonally positioned corners of the unit cube such that $x_0 < x_1, y_0 < y_1$ and $z_0 < z_1$. The curvatures at the corners of the unit cube are $\kappa_{(x_0,y_0,z_0)}, \kappa_{(x_1,y_0,z_0)}, \kappa_{(x_0,y_1,z_0)}, \kappa_{(x_1,y_1,z_0)}, \kappa_{(x_0,y_0,z_1)}, \kappa_{(x_1,y_0,z_1)}, \kappa_{(x_0,y_1,z_1)}$ and $\kappa_{(x_1,y_1,z_1)}$. The differences x_d, y_d and z_d between each of x, y, z and the smallest coordinate (x_0, y_0, z_0) are:

$$\begin{aligned} x_d &= \frac{x - x_0}{x_1 - x_0} \\ y_d &= \frac{y - y_0}{y_1 - y_0} \\ z_d &= \frac{z - z_0}{z_1 - z_0} \end{aligned} \tag{3.11}$$

First interpolate along x-axis, which gives:

$$\begin{aligned} c_{00} &= \kappa_{(x_0,y_0,z_0)}(1 - x_d) + \kappa_{(x_1,y_0,z_0)}x_d \\ c_{10} &= \kappa_{(x_0,y_1,z_0)}(1 - x_d) + \kappa_{(x_1,y_1,z_0)}x_d \\ c_{01} &= \kappa_{(x_0,y_0,z_1)}(1 - x_d) + \kappa_{(x_1,y_0,z_1)}x_d \\ c_{11} &= \kappa_{(x_0,y_1,z_1)}(1 - x_d) + \kappa_{(x_1,y_1,z_1)}x_d \end{aligned} \tag{3.12}$$

Then interpolate the above values along the y-axis:

$$\begin{aligned} c_0 &= c_{00}(1 - y_d) + c_{10}y_d \\ c_1 &= c_{01}(1 - y_d) + c_{11}y_d \end{aligned} \tag{3.13}$$

Finally the curvature at point (x, y, z) is obtained by interpolating the above values along the z-axis:

$$\kappa_{(x,y,z)} = c_0(1 - z_d) + c_1 z_d \tag{3.14}$$

Torsion at an arbitrary point is computed in a similar way.

3.4.2 Velocity Direction Entropy

Xu et al. [92] 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 [92], for each point on a streamline, the entropy of a small neighborhood around that point is computed. Intuitively, the more complicated the neighborhood is, the higher its velocity direction entropy value is. To do so, the 3D vector space needs to be quantized into a certain number of bins, count how many velocity vectors fall into each bin, and compute the entropy by its definition 3.1 (Page 29). For example, for a straight streamline, its velocity direction entropy is zero since all the velocity vectors fall into the same bin. To quantize the 3D vector space, a sphere partition algorithm [46] is leveraged and the number of bins is chosen empirically to be 50. It is 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.

3.4.3 Tortuosity

Tortuosity was first proposed by McLoughlin et al. [57], and it has a low computation cost. Tortuosity measures the degree of deviation from a straight line for a curve. It is calculated as the ratio of the length of curve to the straight line distance between the curve's start and end points. For example, a straight line has the lowest tortuosity value of one, and the tortuosity of a half circle with radius r is $\frac{\pi}{2} = \frac{\pi r}{2r}$. 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 3.2).

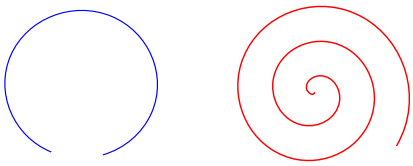


Figure 3.2: Tortuosity vs. velocity direction entropy: both curves have similar velocity direction entropies because their tangent vectors almost point in every direction in a 2D space. However, the red one looks more complicated than the blue one and has a higher tortuosity value.

To compute the tortuosity at point x on a streamline, the following formula is used:

$$T(x) = \frac{\alpha(x)}{\|p(x) - p(0)\|} \tag{3.15}$$

where $\alpha(x)$ is arc-length between the point and the streamline's start point, and $p(x)$ (*resp.*, $p(0)$) is the coordinates of point x (*resp.*, the start point). The arc-length $\alpha(x)$ is calculated by the sum of the lengths of all the straight line segments between the start point and the point x. These straight line segments are determined by the consecutive points (generated by Runge-Kutta method mentioned on Page 3) from the streamline's start point till the point x.

3.5 Results and Discussion

The purpose of this section is to demonstrate the utility of the spatially sensitive bag-of-features in two common flow field exploration tasks: streamline similarity query and streamline clustering. The experiment uses a total of seven flow data sets listed in Table 3.1. The five critical points data set is a synthesized flow field consisting of two spirals, two saddles and one source. The tornado data set is from a simulation of a tornado event. The supernova data set is from a simulation of the explosion of stars. The car flow data set is from the simulation of the air flow around a car. The crayfish data set is from a simulation of the heat flow around a cooking crayfish. 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. Finally, the computer room data set is from a simulation of air flows inside a computer room.

Table 3.1
The timing results of seven flow data sets for feature and spatially sensitive bag-of-features computation.

data set	dimension	initial # lines	average # points per line	feature evaluation time	SS-BoF evaluation time
five critical pts	$51 \times 51 \times 51$	500	60	0.372s	0.245s
tornado	$64 \times 64 \times 64$	500	300	0.833s	0.785s
supernova	$100 \times 100 \times 100$	500	106	0.553s	0.323s
car flow	$368 \times 234 \times 60$	500	338	0.846s	0.711s
crayfish	$322 \times 162 \times 119$	800	354	1.327s	1.422s
solar plume	$126 \times 126 \times 512$	600	492	1.674s	1.799s
computer room	$417 \times 345 \times 60$	800	227	1.28s	1.076s

3.5.1 Configuration and Timing

A hybrid CPU-GPU solution is used for computation with the following hardware configuration: Intel Core i7 quad-core CPU running at 3.20GHz, 24GB main memory and an nVidia GeForce GTX 580 graphics card. Streamline tracing and feature calculation are performed on the GPU using CUDA. The input velocity field was loaded into the texture memory on GPU such that velocity derivatives can be computed efficiently. The spatially sensitive bag-of-features for all the streamlines are computed in parallel using OpenMP. As it can be seen from Table 3.1, the pre-processing including feature and SS-BoF evaluation is finished very quickly even for large data sets such as solar plume and computer room.

3.5.2 Streamline Similarity Query

Due to the large number of streamlines that are displayed, visual cluttering and occlusion become a challenge for flow field exploration. In order to overcome this problem, users can pick a streamline of interest and request to display streamlines similar to the query streamline. When a query streamline is picked, the similarity between that streamline and each of the remaining streamlines is calculated. After that, users can adjust the number of similar streamlines to be displayed. Since the spatially sensitive bag-of-features for each streamline is precomputed, the query can be done in real time. The streamline similarity query results are compared using two similarity metrics: one is the spatially sensitive bag-of-features explained in Section 3.3, and the other one is the similarity metric proposed by Lu et al. [54]. Readers can refer to Section 2.1 (Page 20) on the details of their similarity metric. For the purpose of fair comparison, two similarity metrics are computed using the same set of features described in Section 3.4. Lu et al. determined the number of bins in the 1D feature histogram of each streamline segment using Scott's choice [78]. However, in the following experiment, the number of bins of the feature histogram is set to be the same as the size of the vocabulary used in spatially sensitive bag-of-features so that the two methods can be compared using the same representative feature descriptors.

Figures 3.3 to 3.9 illustrate the comparison between the two methods. In these

figures, the left column shows the query streamlines, the middle column shows the results using spatially sensitive bag-of-features (SS-BoF), and the right column shows the results obtained using the method by Lu et al. [54]. Notice that only the 20% most similar streamlines are displayed. Overall, SS-BoF has a better performance in the sense that the streamlines in the query results look more similar to the query streamline.

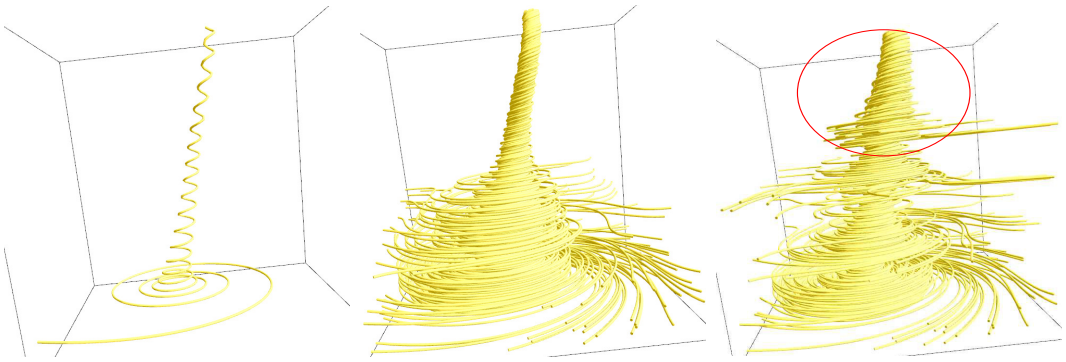
For the five critical point data set in Figure 3.3, the query streamline looks like a straight line. For both similarity measures, the streamlines similar to the query are found at the two corners of the bounding box . However, some spiral streamlines appear in the query results using Lu's similarity metric, which do not look like the query streamline.

(a) Query streamline (b) SS-BoF (c) Lu's similarity metric

Figure 3.3: Query result comparison using the five critical points data set

For the tornado data set in Figure 3.4, it is obvious that similar streamlines found using SS-BoF make more sense. The query streamline has an elongated spiral, which

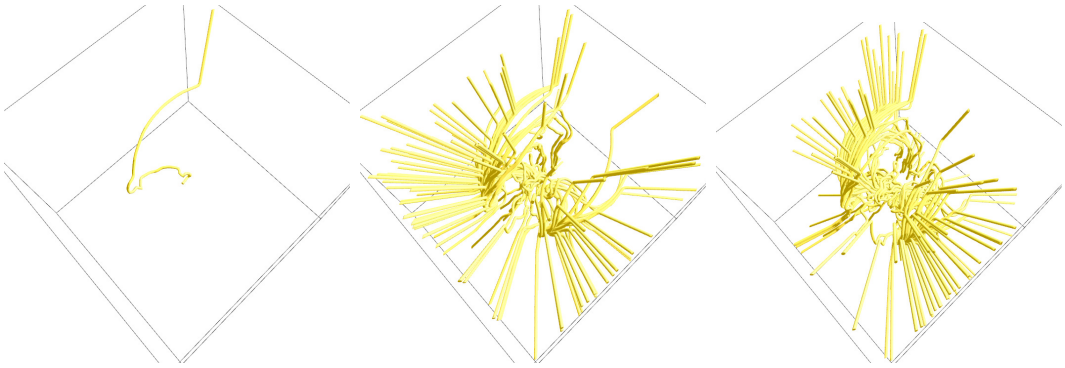
also appears in most of the streamlines found using SS-BoF; however, in the query results found using Lu's similarity metric, there are many streamlines that instead have shorter spirals shown in the red circle.



(a) Query streamline (b) SS-BoF (c) Lu's similarity metric

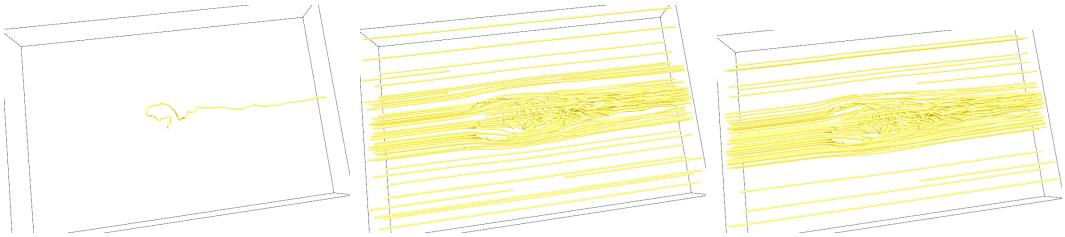
Figure 3.4: Query result comparison using the tornado data set

For the following three data sets in Figures 3.5 to 3.7: supernova, car flow and crayfish, it is hard to say which similarity metric works better because the streamlines found by both metrics look similar enough to the query streamline. The reason for the fact that some straight streamlines show up as similar streamlines in the car flow data set in Figure 3.6 is that there does not exist a sufficient number of streamlines similar to the query streamline given the current threshold on the percentage of similar streamlines to display. Similarly, in the crayfish example in Figure 3.7, some streamlines considered similar to the query have parts that are close to straight lines, even though the query streamline does not contain any straight part. Again this is because there are not enough streamlines in the crayfish data set which look close enough to the query.



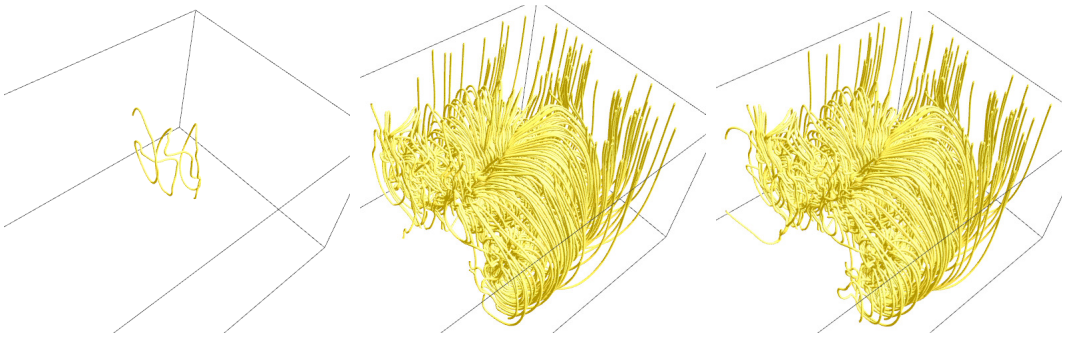
(a) Query streamline (b) SS-BoF (c) Lu's similarity metric

Figure 3.5: Query result comparison using the supernova data set



(a) Query streamline (b) SS-BoF (c) Lu's similarity metric

Figure 3.6: Query result comparison using the car flow data set



(a) Query streamline (b) SS-BoF (c) Lu's similarity metric

Figure 3.7: Query result comparison using the crayfish data set

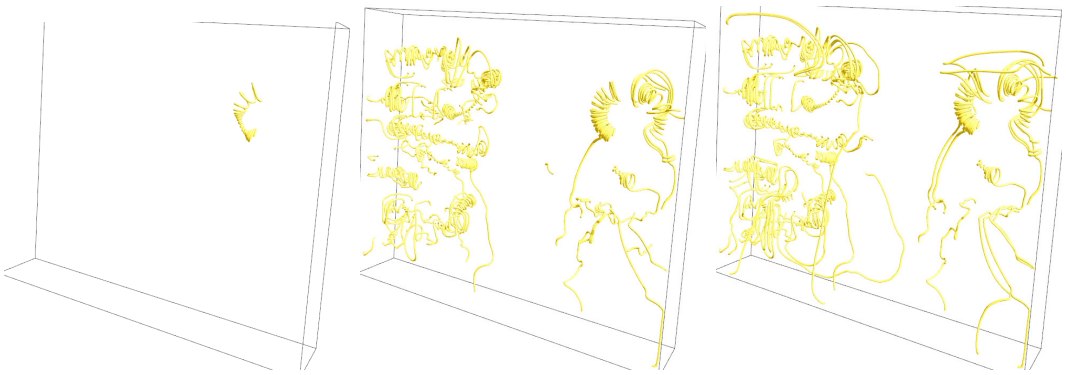
For the plume data set in Figure 3.8, the query streamline has a big 'U' shape in it. Many of the similar streamlines found using SS-BoF also have the 'U' shape. However, using Lu's similarity metric gives some streamlines with a small hook (in

red circles) in the results. Therefore, the query results using SS-BoF look better on the plume data set.

(a) Query streamline (b) SS-BoF (c) Lu's similarity metric

Figure 3.8: Query result comparison using the solar plume data set

Finally, for the crayfish data set in Figure 3.9, the two similarity metrics lead to very similar query results. Again, the difference is too small to determine which metric is better.



(a) Query streamline (b) SS-BoF (c) Lu's similarity metric

Figure 3.9: Query result comparison using the computer room data set

3.5.3 Streamline Clustering

Streamline clustering is an effective way of visualizing a large number of streamlines by grouping together the streamlines with similar shapes. A key step in common clustering algorithms is to compute the similarity between two elements. Better similarity metric will lead to better clustering results. The following experiment will apply *affinity propagation* to the same set of streamlines twice, each time with a different similarity metric, so that clustering results can be compared subjectively to determine which similarity metric is better.

The experiment uses affinity propagation [32] as the clustering algorithm. Affinity propagation is a clustering algorithm which can automatically determine the best number of clusters by message passing. The number of clusters is affected by the value of *preference*. Low preferences leads to a small number of clusters, whereas high preferences lead to a large number of clusters. The input to the affinity propagation algorithm is a symmetric similarity matrix where each matrix element at (i, j) is the similarity between data points i and j. In the following experiment, two similarity matrices are computed for the purpose of comparison, first based on SS-BoF and then Lu's similarity metric.

The experiment is performed using the solar plume (from real simulation) and the

tornado (computer synthesized) data sets. The results are illustrated in Figures 3.10 to 3.12. Ideally, the numbers of generated clusters should be kept the same for fair comparison; however, since affinity propagation does not allow users to specify the number of clusters, best efforts are made to make the number of clusters as close as possible by adjusting the value of preferences mentioned in the previous paragraph.

For the plume data set in Figure 3.10, five clusters are generated when SS-BoF is used as the similarity metric, whereas only four clusters are produced using Lu's similarity metric. It can be seen that streamlines with different shapes are better separated with the use of SS-BoF. For example, the streamlines in the red cluster (top row) do not look like the streamlines in any other cluster, and thus consist of a separate cluster; however, these streamlines are merged into the blue cluster (bottom row) using Lu's similarity metric. Moreover, both the yellow and the blue clusters (bottom row) contain similar streamlines because they all have a big 'U' shape, so these similar streamlines are better to be in the same cluster. Notice how the clustering using SS-BoF successfully groups these streamlines into the purple cluster (top row). Finally, the blue cluster (bottom row) contains some streamlines with a small hook (in the center of the vector field) which do not look like the remaining streamlines in the same cluster. These streamlines appear in the yellow cluster (top row) using SS-BoF, and they do look similar to the rest of the streamlines in the same cluster.

For the tornado data set in Figures 3.11 and 3.12, it is obvious that the streamlines

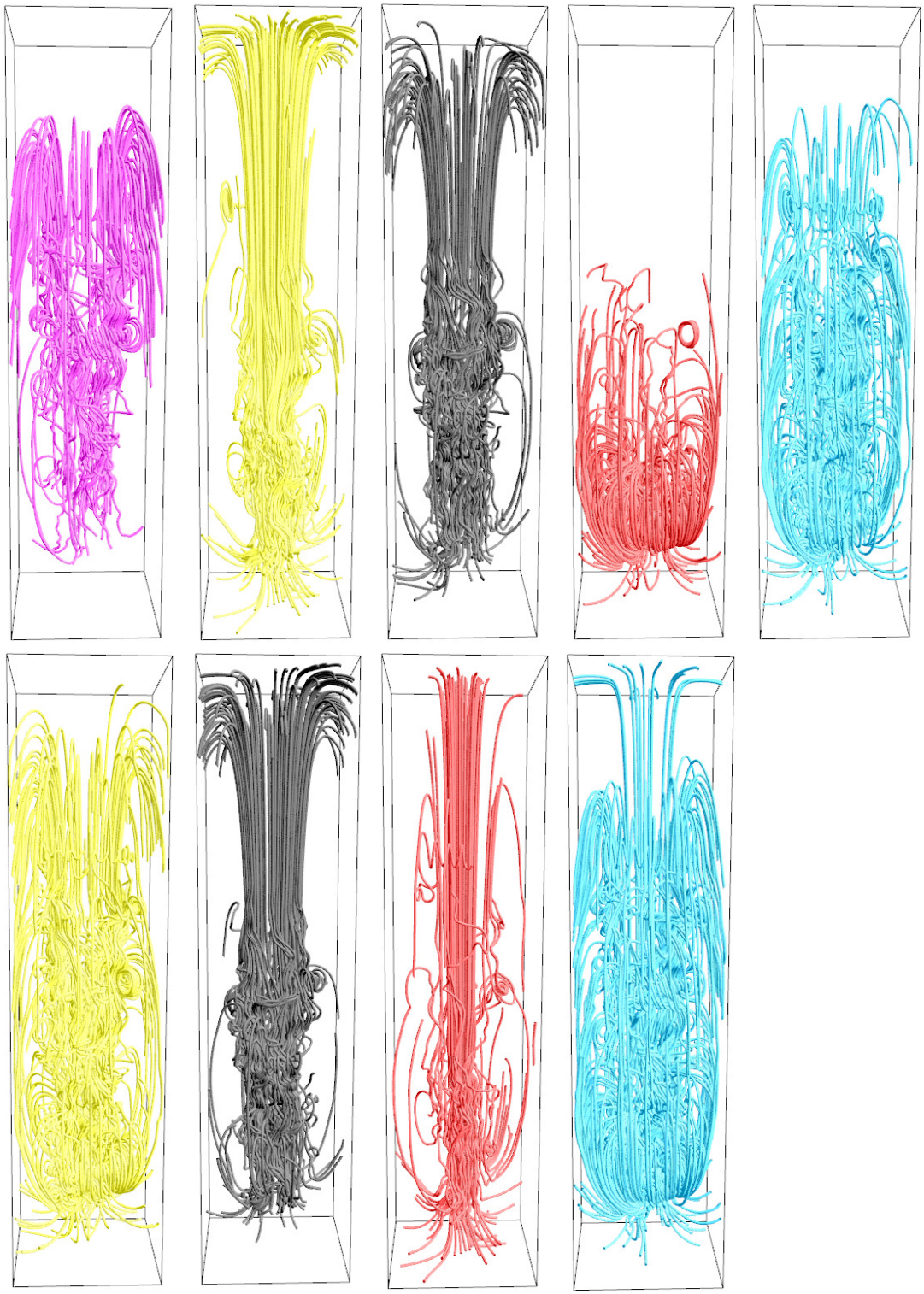


Figure 3.10: Clustering results for solar plume data set using SS-BoF (top) and the measure based on Lu et al. [54] (bottom).

looking very different appear in the same cluster when Lu's similarity metric is employed by the clustering algorithm. For example, in the red cluster in Figure 3.12, there are some streamlines at the top of the bounding box which do not look like spirals. Also, the yellow cluster in Figure 3.12 has both long and short spirals. However, this is not the case when SS-BoF is used. Notice that in Figure 3.11, the longer spirals appear in the red and blue clusters, whereas the shorter spirals are in the yellow and black clusters.

There does exist a problem with SS-BoF. In the purple cluster in Figure 3.11, there is a long spiral which does not look like the remaining streamlines in the same cluster. When this long spiral is compared with the streamlines in both the red and the blue clusters in Figure 3.11, it does not look like them either because (1) it does not have the flat part at its bottom, and (2) the top part of the streamline (circled in green) is twisting up very fast, which implies large torsion and small curvature; however, this is not the case for the spirals in the red or the blue cluster in Figure 3.11. The clustering algorithm fails to make a separate cluster for this single streamline because the distance between its SS-BoF and the SS-BoF of the other streamlines in the same cluster is not large enough. This suggests that the current features (Section 3.4) may not be sufficient to describe the difference.

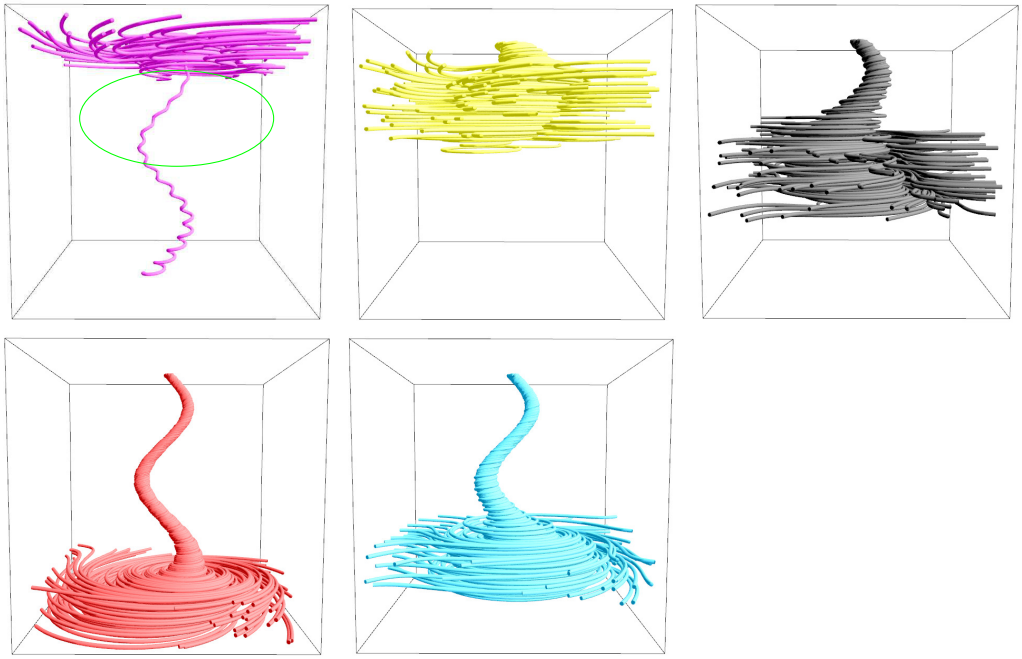


Figure 3.11: Clustering results for tornado data set using SS-BoF.

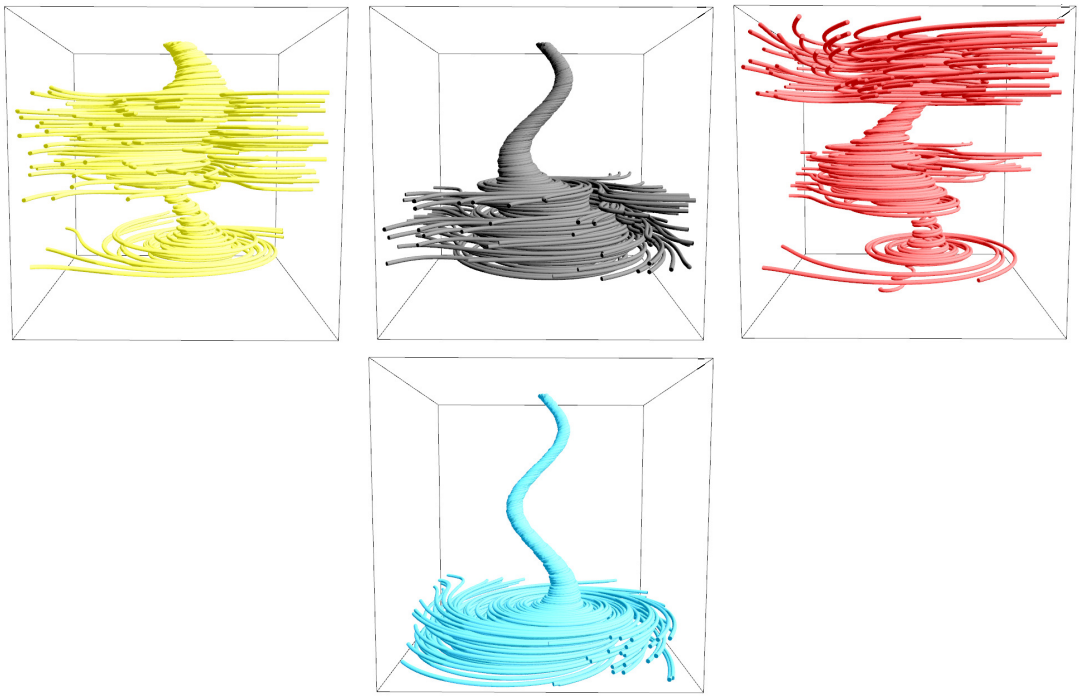


Figure 3.12: Clustering results for tornado data set using Lu et al. [54].

3.6 Conclusion

This chapter presents a novel method of measuring streamline similarity based on spatially sensitive bag-of-features. Each streamline is represented by its own spatially sensitive bag-of-features, which encodes the statistical distribution of different features on the streamline and the spatial relationship among the features. Only four types of features are considered currently: curvature, torsion, tortuosity and velocity direction entropy; however, other features can be incorporated easily into the computation of spatially sensitive bag-of-features by just adding them into the feature descriptors. The weighted Manhattan distance is used to measure the distance between two spatially sensitive bag-of-features representations.

Although this approach works reasonably well as demonstrated in Section 3.5, two things deserve further study:

- Search for new features to add into spatially sensitive bag-of-features. Streamlines have far less features than images. For images, besides SIFT and MSER mentioned in Section 3.2.3, researchers have designed many other image features [88] for various computer vision tasks. Since streamlines are essentially space curves, it is challenging to find out other features independent from curvature and torsion which can also describe a curve's shape characteristics well.

In other words, the new features should be able to characterize the shape of a streamline globally since curvature and torsion are already the local features.

- Design better distance measure for evaluating the difference between two spatially sensitive bag-of-features (Section 3.3). Weighted Manhattan distance may not work well sometimes as shown in Section 3.5.3. An interesting direction is to study how to leverage machine learning to obtain a better distance measure. One possible way to do this is through the machine learning based ranking algorithm. Learning to rank [50] has been widely used in information retrieval tasks such as web search: given a query, the list of documents ranked in decreasing order of relevance is returned. In order to learn such a ranking function, the training data should be provided by users and contain the following: a set of query streamlines, and, for each query streamline, a set of similar streamlines along with the labels denoting the level of similarity to the query. Note that users can also include a set of dissimilar streamlines in the training data. With such a ranking function, the result of streamline query can be shown based on the level of similarity.

Chapter 4

Extracting Flow Features via Supervised Streamline Segmentation

4.1 Overview

The previous chapter presents two ways (i.e., streamline query and clustering) of reducing visual clutter for the vector field visualized with a large amount of streamlines. Sometimes, however, those two methods may not be enough because the interesting features which correspond to partial streamlines (i.e., a part of a streamline) may still

be occluded after clustering. This chapter discusses an approach which allows partial streamlines to be extracted, and these partial streamlines are grouped by shape similarity for feature exploration. Each group of partial streamline corresponds to some feature which users are interested to see. Since there is no rigorous definition of features, users will be asked to provide some examples of the features specific to their applications. This can be done by manually segmenting a few representative streamlines into segments corresponding to different features. The remaining streamlines from the same vector field will be segmented based on those user-defined features in the hope that the features will also be extracted if they exist in the remaining streamlines. This whole process is called supervised streamline segmentation because *supervised learning* is applied to achieve the goal. Figure 4.1 illustrates the process of extracting user-defined flow features.

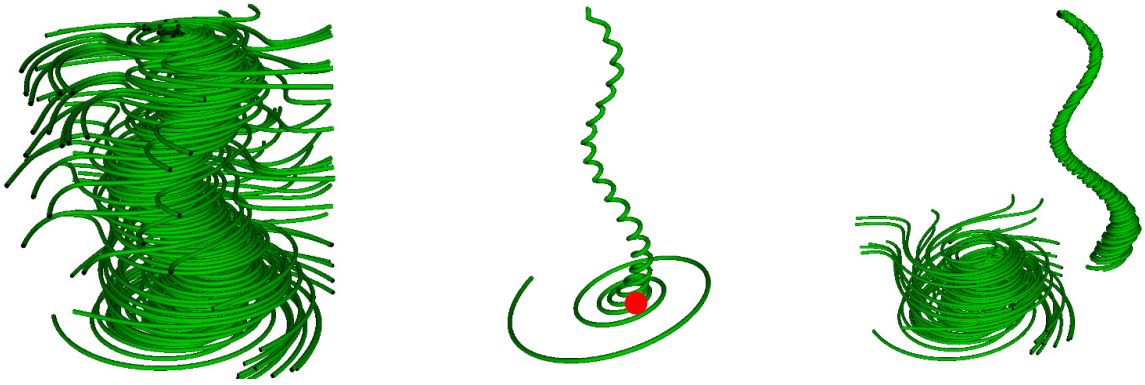


Figure 4.1: Given an input pool of streamlines (left), each streamline is segmented using a learned classifier (Section 4.2) for segmentation points (middle, the red point is the segmentation point found by our algorithm). Partial streamline features specified by users will be clustered based on their similarities (right).

In the following, Section 4.2 introduces some background on supervised learning and explains the details of *support vector machine*. Section 4.3 explains in detail how supervised streamline segmentation is performed. Section 4.4 discusses the results of applying this technique to different flow data sets and parameter selection. Section 4.5 compares this technique with other state-of-the-art feature extraction methods. Finally, Section 4.6 has the conclusions and discusses some future work. This work has been published by the international journal of *Computers & Graphics* [49].

4.2 Supervised Learning and Support Vector Machine

4.2.1 Supervised Learning

Supervised learning is a machine learning task of inferring a function from labeled *training data*. The training data consists of a set of *training examples*. In supervised learning, each example is a pair consisting of an input object (typically a vector consisting of different features of the object) and a desired output value. A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new input object to an output value. *Training* refers to the process during which the precise form of the inferred function is determined. If the

possible output values consist of a set of discrete values, the mapping process is called *classification*. So classification is the problem of identifying which category a new input object belongs, on the basis of training examples. If there are only two possible output values, the classification is called *binary classification*. An example of binary classification would be assigning a given email into spam or non-spam class. An algorithm which implements classification is called a *classifier*. For a binary classification problem, *positive examples* are pairs $(\mathbf{x}, y)$ where $\mathbf{x}$ (boldface indicates vectors) is an input object and y is the correct categorization of $\mathbf{x}$, and *negative examples* are pairs of $(\mathbf{x}, y)$ where y is an incorrect categorization of $\mathbf{x}$. *Training error* is the fraction of training examples a classifier misclassifies. The ability to categorize correctly unseen input objects is known as *generalization*.

4.2.2 Support Vector Machine

A support vector machine (SVM) [76] is a supervised learning model. In the case of binary classification, an SVM constructs a hyperplane which can be used for classification. If a hyperplane can be found to completely separate positive examples from negative examples, the training examples are called *linearly separable*. To determine the label of a new input object, an SVM simply needs to figure out which side of the hyperplane the input objects falls onto. It can be proved that 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*. More formally, for linearly separable training examples, an SVM learns a linear model of the form $y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$ for binary classification problems, where $\mathbf{w}^T$ and b are the normal and the intercept of a hyperplane, and $\mathbf{w}^T\mathbf{x}$ is the inner product between $\mathbf{w}^T$ and $\mathbf{x}$. The corresponding decision function is $f(\mathbf{x}) = \text{sgn}(\mathbf{w}^T\mathbf{x} + b)$, where sgn() outputs 1 if $\mathbf{x}$ is above or on the hyperplane and -1 if x is below the hyperplane.

To construct the optimal hyperplane separating m training examples, the following optimization problem needs to be solved [76]:

$$\begin{aligned} &\underset{b\in\mathbb{R}}{\text{minimize}} \quad \tau(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 \\ &\text{subject to} \quad y^i(\mathbf{w}^T\mathbf{x}^i + b) \geq 1 \text{ for all } i = 1, \ldots, m \end{aligned} \tag{4.1}$$

where $(\mathbf{x}^i, y^i)$ represents the i-th training example.

The above optimization problem is usually solved by its *dual* problem [76]:

$$\begin{aligned} &\underset{\alpha\in\mathbb{R}^m}{\text{maximize}} \quad W(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y^i y^j \langle \mathbf{x}^i, \mathbf{x}^j\rangle \\ &\text{subject to} \quad \alpha_i \geq 0 \text{ for all } i = 1, \ldots, m \text{ and } \sum_{i=1}^{m}\alpha_i y^i = 0 \end{aligned} \tag{4.2}$$

For non-linearly separable training examples, a straightforward approach is to transform them into some high dimensional space where the training examples become linearly separable. However, in order to save computation cost, a *kernel function* is applied to each pair of training examples, which is equivalent to explicitly transforming the training examples into a high dimensional space and then computing the inner product between the transformed training examples. The decision function now takes the form:

$$f(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^{m} y^i \alpha_i k(\mathbf{x}, \mathbf{x}^i) + b\right) \tag{4.3}$$

where $k(\mathbf{x}, \mathbf{x}^i)$ is a kernel function. Specifically, given a mapping ψ which maps the training examples into a high dimensional space, its corresponding kernel function is $k(\mathbf{x}, \mathbf{z}) = \psi(\mathbf{x})^T \psi(\mathbf{z})$. A commonly-used kernel function is the radial basis function (RBF) $\exp(-\|\mathbf{x} - \mathbf{z}\|^2 / 2\sigma^2)$, where σ is a free parameter controlled by users.

In practice, a separating hyperplane may not exist if a high noise level in the training data causes a large overlap of the classes. To allow for the possibility of examples violating Equation 4.1, the *slack variables* $\epsilon_i \geq 0$ for all $i = 1, \ldots, m$ are introduced in order to relax the constraints to

$$y^i(\mathbf{w}^T \mathbf{x}^i + b) \geq 1 - \epsilon_i \text{ for all } i = 1, \ldots, m \tag{4.4}$$

Note that if $\epsilon_i \geq 1$, it means the training example $\mathbf{x}^i$ is misclassified.

A classifier that generalizes well is then found by controlling both the classifier capability (via $\|\mathbf{w}\|$) and the sum of slack variables $\sum_i \epsilon_i$. The goal now becomes minimizing the following objective function:

$$\tau(\mathbf{w}, \epsilon) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \epsilon_i \tag{4.5}$$

subjects to the constraint (Equation 4.4), where the constant $C > 0$ determines the trade-off between maximizing classifier's ability of generalization and minimizing training error.

4.2.3 A Guide to libSVM

libSVM [2] is a popular library for support vector machine. It suggests that *cross-validation* and *grid-search* should be done in order to get a classifier with a reasonably good generalization capability. In v-fold cross-validation, the training set is first divided into v subsets of equal size. Sequentially one subset is tested using the classifier trained on the remaining $v - 1$ subsets. Thus, each instance of the whole training set is predicted once so the cross-validation accuracy is the percentage of the data which are correctly classified.

If RBF kernel and slack variables are used, there are two free parameters which users can adjust: C and σ. It is not known beforehand which C and σ are best for a

given problem. Therefore, libSVM recommends a grid-search on them using cross validation. The goal is to identify a good (C, σ) so that the classifier can accurately classify unseen objects. Grid-search tries exponentially growing sequences of C and σ to identity good parameters for a given problem. For example, $C = \{2^{-5}, 2^{-3}, \ldots, 2^{15}\}$ and $\sigma = \{2^{-15}, 2^{-13}, \ldots, 2^{3}\}$.

4.3 Supervised Streamline Segmentation

A user-guided streamline segmentation framework is illustrated in Figure 4.2. For each data set, users are required to manually segment only a small number of streamlines in 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 an SVM to obtain a binary classier. Finally, the classifier is used 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. The training process (the dashed line in Figure 4.2) can be repeated if users are not satisfied with the segmentation results.

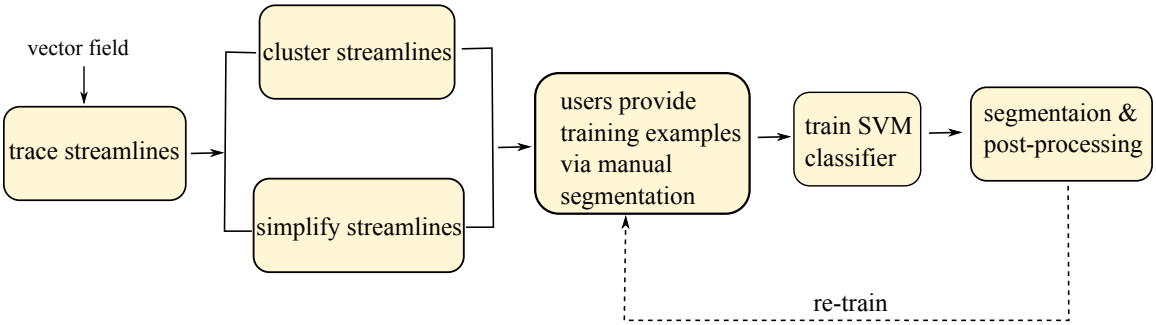


Figure 4.2: The supervised streamline segmentation framework

There are two subtle issues to solve during the training process. First, only representative streamlines should be presented to users for manual segmentation because this will save users the work of segmenting similar streamlines. Therefore, streamline clustering is performed to automatically select representative streamlines for users. Second, since a streamline is usually traced using a small time step (Chapter 1), the points on a streamline are very close to each other. If all of the points are visualized at the same time, it will be too cluttered and hard for users to pick individual points as segmentation points. This is the reason why streamline simplification is performed. Later in this section, another benefit of streamline simplification for training will also be explained. In the following, Section 4.3.1 discusses the features used for supervised learning, Section 4.3.2 explains how training examples are collected, Section 4.3.3 gives the details of the training process, and finally Section 4.3.4 introduces the segmentation algorithm and post-processing step.

4.3.1 Features vectors

In machine learning, an input object is usually represented by a *feature vector* which is an n-dimensional vector of numerical features, so an important task is to determine which features should be used. Since the goal is to obtain a classifier which decides whether a point is a segmentation point, the chosen features should be able to capture the important characteristics of a segmentation point. Intuitively, because the streamline segments on both sides of a segmentation point should "look" differently, a feature vector should include the metrics which evaluate the similarity between two neighboring segments. Note that the neighborhood size plays an important role in the similarity comparison, which is illustrated in Figure 4.3. In this figure, assume the goal is to determine whether the blue point is a segmentation point. The answer seems to be affirmative if the two neighboring segments with green end points are compared because one of them is straight and the other is wavy. However, if the neighborhood size is made larger such that the two segments with red end points are compared, the answer may become less affirmative. Because it is not known what features are contained in a streamline, it is nearly impossible to choose an appropriate neighborhood size in advance, and hence different neighborhood sizes are considered. The same feature will be computed based on these different neighborhood sizes, and the resulting feature vectors are called multiscale feature vectors. More specifically, 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 are compared. Notice that a fixed number of streamline points for neighborhood size is not preferred because different streamlines in the same flow field may have very different numbers of points.

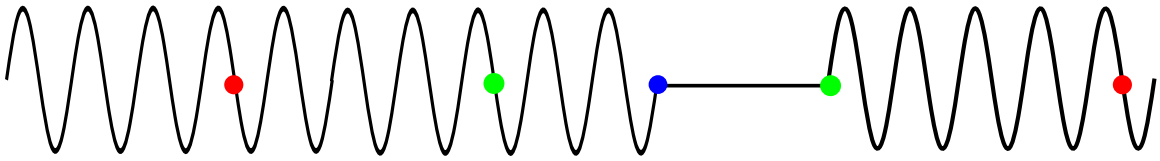


Figure 4.3: Importance of neighborhood size: the blue point may be considered as a segmentation point if the two neighboring segments with green end points are compared but not if a larger neighborhood size is considered (marked by red points).

The following metrics for comparing neighboring segments are used: (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, each of these metrics will be explained in detail.

4.3.1.1 Velocity direction entropy and Tortuosity

Similar to Section 3.4.2, the entropy for a streamline segment is used as an indicator of its complexity. After the velocity direction entropy is computed for each of the two neighboring segments around a point, the ratio of the smaller entropy value to the larger one is calculated, and incorporated into that point's feature vector. Intuitively, the smaller the ratio is, the more likely that point is a segmentation point.

The ratio of tortuosities (Section 3.4.3) of two neighboring segments is also included in a feature vector. The tortuosity ratio is computed in the same way as velocity direction entropy ratio, thus making the ratio between zero and one.

4.3.1.2 Curvature and torsion histogram

Section 3.4.1 introduces a way of computing curvature and torsion: first compute them at grid points and then use the tri-linear interpolation to find their values at an arbitrary point. Using this way, the ranges of curvature/torsion values of different vector fields are usually different because the ranges of the magnitudes of the input velocities are varied. If the training examples are collected from different vector fields, a potential problem for SVM may arise since the best accuracy of SVM is achieved if the same components across different training examples are in the same range [2].

Therefore, the way of computing discrete curvature and torsion as described in [12] is adopted. Using this method, it is guaranteed that the range of curvature (*resp.*, torsion) will be $[0, \pi)$ (*resp.*, $(-\pi, \pi]$) . This method starts by defining an orthonormal frame X_i, Y_i, Z_i at each streamline point p_{i+1}:

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

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 κ_i at each point p_i and the discrete torsion τ_i for each segment $p_i p_{i+1}$ are given by the following formulas:

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

In order to obtain a histogram describing the shape characteristics of a streamline, curvature and torsion histograms are first computed, and the two histograms are concatenated into a single 1D histogram similar to [54]. The number of bins are empirically set to 20 for curvature histograms and 40 for torsion histograms because the discrete curvature and torsion have different ranges. By doing this, each bin corresponds to $180\,^\circ/20 = 9\,^\circ$. This granularity is enough for the purpose of similarity comparison because even though the numbers of bins are increased to 80 and 160 for curvature and torsion histograms respectively, no noticeable difference is observed regarding the clustering results later (Section 4.3.2.1). Finally, the 1D histogram is normalized to make it scale-invariant.

The distance between two 1D histograms is measured using the earth mover's distance (EMD) [72], which is a measure of the distance between two probability distributions. Intuitively, given two distributions, one can be seen as a mass of earth properly spread in space, the other as a collection of holes in that same space. Then, the EMD

measures the least amount of work needed to fill the holes with earth. Computing the EMD is based on a solution to the well-known transportation problem [36]. Suppose that several suppliers, each with a given amount of goods, are required to supply several consumers, each with a given limited capacity. For each supplier-consumer pair, the cost of transporting a single unit of goods is given. The transportation problem is then to find a least-expensive flow of goods from the suppliers to the consumers that satisfies the consumers' demand. As Lu et al. [54] pointed out, the EMD distance can be approximated by the L_1-distance between two 1D cumulative histograms to reduce computation cost:

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

where P_{cdf} and Q_{cdf} are the cumulative distribution functions of the two normalized histograms, and n is the number of bins in each histogram.

4.3.1.3 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, it is observed that the following ratio should be small (e.g., less

than 1.0):

$$\frac{V(E_{s_1}) + V(E_{s_2})}{V(E_{s_1 \cup s_2})} \tag{4.6}$$

where V measures the volume of an ellipsoid.

For example, in Figure 4.4 (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. 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. This heuristic is also a good indicator for separating 3D features when there is an abrupt change in torsion even though those features have similar complexity. Take Figure 4.4 (b) for an example. 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 4.4.2 will discuss in detail the impact of this heuristic on the classifier performance and final segmentation

results.

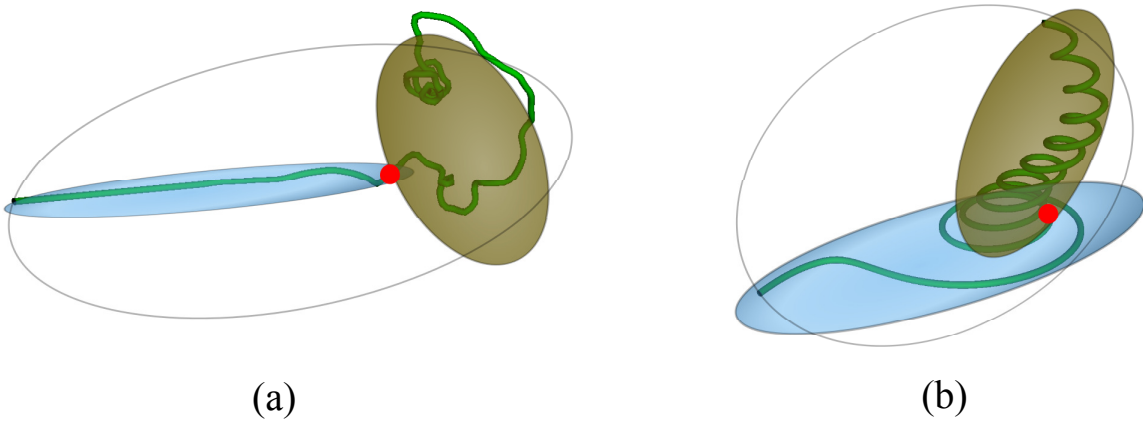


Figure 4.4: The blue and the brown ellipsoids are the minimum volume ellipsoids bounding the streamline segments on two sides of the red point. The minimum volume bounding ellipsoid for the whole streamline is shown as white ellipsoids.

The algorithm proposed by Kumar et al. [44] is used to compute minimum volume bounding ellipsoids. At first sight, principle component analysis (PCA) [40] seems to be a good way of computing approximate bounding ellipsoids. However, some experiments showed that PCA does not perform as well as Kumar's method. There are other bounding shapes such as cubes or spheres, but ellipsoids can bound a streamline more compactly than other shapes.

4.3.2 Training examples collection

In order to obtain a classifier which determines whether a point is a segmentation point or not, training examples which include both segmentation and non-segmentation

points need to be collected. Each of these training examples consists of a feature vector computed based on the features mentioned in the previous section and a label indicating whether it is a segmentation point. Since users need to provide the training examples through manual segmentation, the question of how to reduce their work and to obtain an accurate classifier at the same time needs to be addressed.

There are two tasks involved during training examples collection: (1) choose streamlines for manual segmentation, and (2) pick the segmentation points on a streamline. For the first task, it would be difficult for users to do because they may not be able to pick any streamline they want due to occlusion, and even though they could, they have to keep track of the streamlines they have manually segmented to avoid segmenting similar streamlines. For the second task, it is better to only have users specify the segmentation points (i.e., positive examples) but not non-segmentation points (i.e., negative examples) to reduce their workload. Another issue during segmentation point picking is that, due to the small time step used in tracing streamlines, the points on a streamline are usually too close to be easily picked individually.

To address the above issues, some representative streamlines are automatically picked out for users to segment (Section 4.3.2.1), and streamlines are simplified (Section 4.3.2.2) such that users can pick individual points easily. Users only need to pick segmentation points as positive examples, and the remaining points will be used as negative examples.

4.3.2.1 Automatically picking streamlines for training

The streamlines are clustered based on their similarity such that users only need to choose the streamlines for manual segmentation from the cluster representatives. In order to make the clustering fully automatic without users having to specify the number of clusters, affinity propagation [32] is used for clustering (Section 3.5.3).

Affinity propagation requires the distances of different streamline pairs as input. To compute the distance between two streamlines, the 1D histogram (Section 4.3.1.2) is constructed for each streamline such that the distance can be computed as an approximate EMD distance (Section 4.3.1.2) between the two histograms. Notice that the similarity values used as input to affinity propagation should be the negative of the approximate EMD as required by affinity propagation. Figure 4.5 shows an example of the clustering results of the tornado data set (Section 4.4.1), where six clusters ((b)-(g)) are generated. The cluster representatives are shown in green.

Note that it is not necessary to achieve the "best" clustering results in this step because users are allowed to provide more training examples and re-train the segmentation point classifier later.

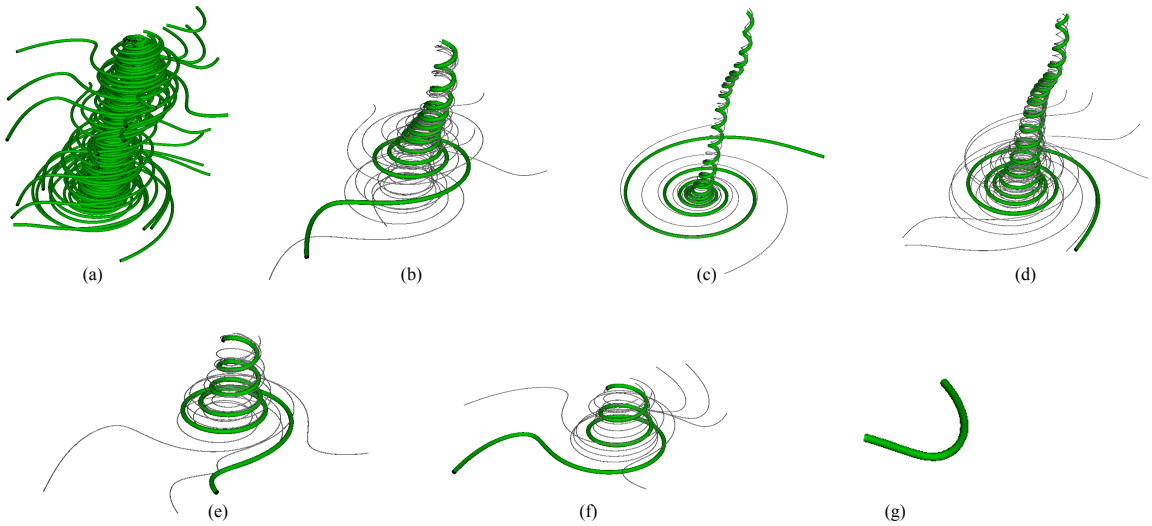


Figure 4.5: Streamline clusters and their representatives: the input streamlines of the tornado data set are shown in (a). After applying affinity propagation, six clusters ((b)-(g)) are obtained and cluster representatives are shown in green.

4.3.2.2 Generating training examples

Given a streamline, users need to pick the points where they want to segment it. For each segmentation point, the velocity direction entropy ratio, tortuosity ratio, histogram difference, and volume ratio of minimum bounding ellipsoids using different neighborhood sizes (Section 4.3.1) are computed, and these values consist of the feature vector for the positive example. For points not picked by users, they are considered as non-segmentation points. The same features will be computed for each non-segmentation point and used as the feature vector for the negative example.

Instead of presenting all the points on a streamline to users, a curve simplification

algorithm [3] is applied to reduce the number of candidates for them to choose segmentation points from. There are a couple of reasons for doing curve simplification:

- Make it easy to collect training examples. A large number of points are generated during streamline tracing in order for streamlines to have better visual quality, which makes it difficult for users to pick individual points because the points are too close to each other.

- 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, fewer points need to be tested for segmentation points. Also the cost of training will be reduced since there will be less training examples.

A popular curve simplification algorithm [3] approximates a polygonal curve P under the Fréchet error by another polygonal curve P' whose vertices are a subset of the vertices of P. Intuitively, the Fréchet distance between two curves is the minimum length of a leash required to connect a dog and its owner, constrained on two separate paths, as they walk without backtracking along their respective curves from one

endpoint to the other. More formally, a parameterized curve in $\mathbb{R}^d$ can be represented as a continuous function $f : [0,1] \rightarrow \mathbb{R}^d$. A monotonic reparameterization α is a continuous non-decreasing function $\alpha : [0,1] \rightarrow [0,1]$ with $\alpha(0) = 0$ and $\alpha(1) = 1$. Given two curves $f, g : [0,1] \rightarrow \mathbb{R}^d$, the Fréchet distance $\delta_F(f,g)$ is defined as

$$\delta_F(f,g) = \inf_{\alpha,\beta}(\max_{t\in[0,1]} d(f(\alpha(t)), g(\beta(t))) \tag{4.7}$$

where inf stands for the greatest lower bound, $d(x,y)$ denotes the Euclidean distance between points x and y, and α and β range over all monotonic reparameterizations.

Figure 4.6 compares the simplification results using different Fréchet errors, which are measured by the Fréchet distance between the original curve and its simplified one. Originally, the streamline has 147 points. After simplification, 25/18/15 points are left if Fréchet error is set to 0.5/1.0/1.5. In this dissertation, the default Fréchet error is chosen to be 1.0 based on the following two reasons. First, this value is conservative enough (i.e., no over-simplification) such that all the candidate segmentation points are in the simplified streamlines. Second, 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 may be very close to each other. The next section will discuss how negative training examples are generated. The Fréchet error can also be made adjustable in case the default value does not work well for users.

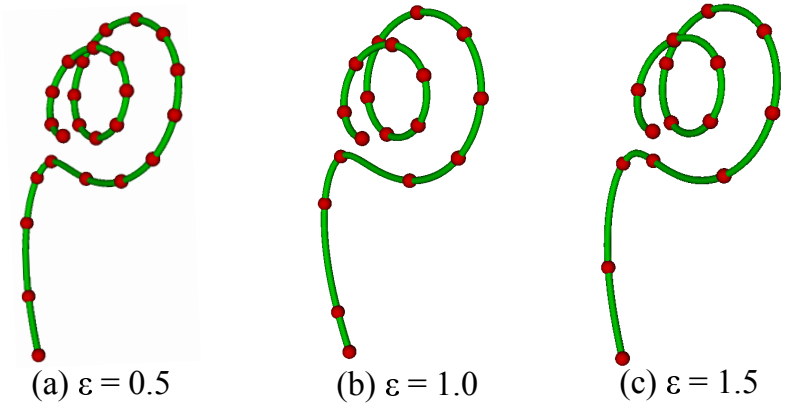


Figure 4.6: A streamline with 147 points is simplified with different Fréchet error ϵ (points left after simplification are shown in red): (a) $\epsilon = 0.5$, 25 points left (b) $\epsilon = 1.0$, 18 points left (c) $\epsilon = 1.5$, 15 points left.

Since determining where to segment a streamline is often not a clear-cut decision, users are allowed 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 nearby points to indicate where segmentation occurs. This also helps alleviate the problem of highly imbalanced training examples (Section 4.3.3) since more positive examples are generated.

4.3.3 Training

Once all the training examples are collected, the library libSVM [21] can be leveraged to train a classifier for detecting segmentation points. However, there still exists a big challenge: highly imbalanced training examples. In other words, there exists far more negative examples than positive ones. This is caused by the fact that only a

few points on a streamline are segmentation points while the remaining ones are all non-segmentation points. libSVM will generate a classifier with a high false negative rate if applied directly to such training data.

To address the above issue, the method described in [83] is used. The method incrementally removes redundant negative examples until the classification performance cannot be improved. Figure 4.7 illustrates this process. In each iteration i, an SVM classifier with RBF kernel is obtained from the training examples $Tr(i)$. If the classification performance of the SVM decreases compared with that from the previous iteration, the SVM with RBF kernel from the previous iteration is considered as the final classifier. Nonlinear SVMs (i.e., SVMs with non-linear kernels) are used because they outperform linear ones on all the training data used in this dissertation. This suggests that the training data are not linearly separable. If the classification performance of the SVM classifier with RBF kernel improves in iteration i, a linear SVM is trained on $Tr(i)$, and the negative support vectors obtained from the training are removed from $Tr(i)$ to form the new training examples $Tr(i+1)$. The linear SVM is chosen over nonlinear SVMs because the training time can be reduced. Notice that the goal here is to remove redundant even noisy negative examples instead of achieving the best training accuracy, some experiments showed that the linear SVM serves this purpose well.

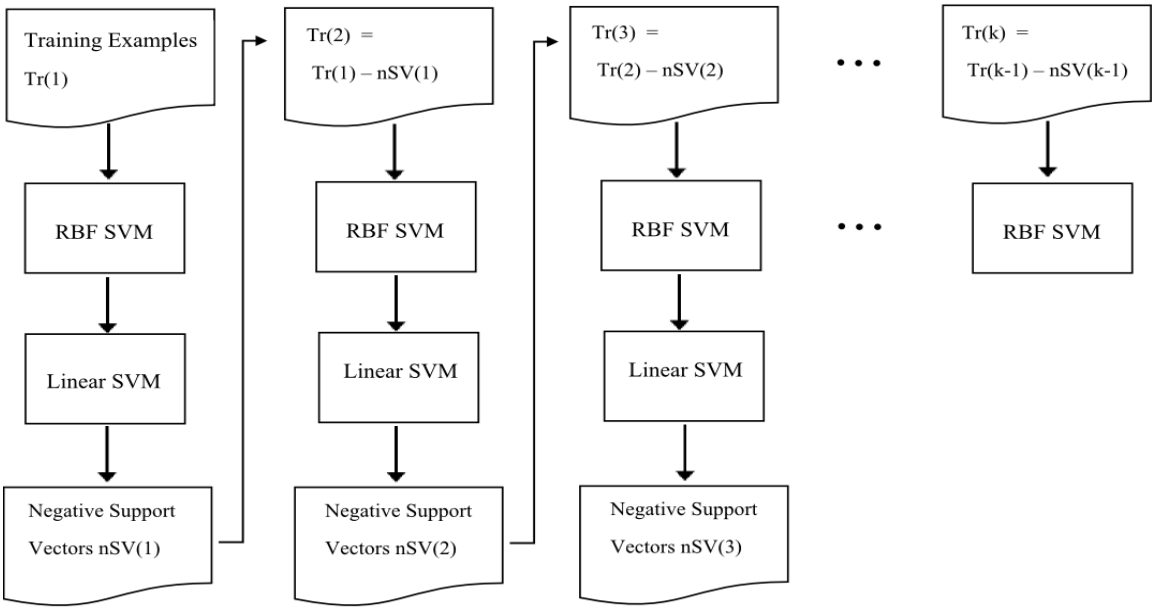


Figure 4.7: Remove redundant negative examples until the classification performance cannot be improved.

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 there are 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. To address this issue, the *receiver operating characteristics (ROC) curve* [29] is introduced, which focuses on the relative tradeoffs between benefits (true positives) and costs (false positives) instead of just the number of correctly classified examples. A binary classifier usually yields a probability which is a numerical value that represents the degree to which an instance is a member of a class. The decision is then

made through thresholding: if the probability is above the threshold, the classifier outputs a positive answer, else a negative one. Each threshold value corresponds to a pair of true positive and false positive values. A ROC curve is the curve with the threshold value on the x-axis and the corresponding output from the classifier on the y-axis. 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.

A method described in [83] can be used to handle highly imbalanced training data for training SVM classifiers. Its pseudo-code 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 (Section 4.2.2) on *aggregation* and check if performance is improved (Lines 8-13).

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.nSV ← linearModel.nSV
7:         examples.Erase(linearModel.nSV)
8:         segPointClassifier ←RBFSVM(aggregation)
9:         auc ←ComputeAUC(segPointClassifer)
10:        if auc ≤ bestAUC then
11:            break
12:        else
13:            bestAUC ← auc
```

4.3.4 Segmentation and post-processing

Streamline segmentation is straightforward once the above classifier is obtained. For each streamline, each point on the simplified streamline is tested for whether being a segmentation point. It is possible that several nearby points are all classified as segmentation points (Figure 4.8 (a)). Therefore, those nearby points need to be grouped and one of them will be picked as a segmentation point.

The goal is to group segmentation points which are close to each other in terms of arclength. In order to achieve this grouping, half of the mean of all the segments' lengths is used as threshold T_S and the segmentation points between which the arclength is less than T_S are grouped together. The same strategy was also adopted by [4] to group salient points of a 3D mesh.

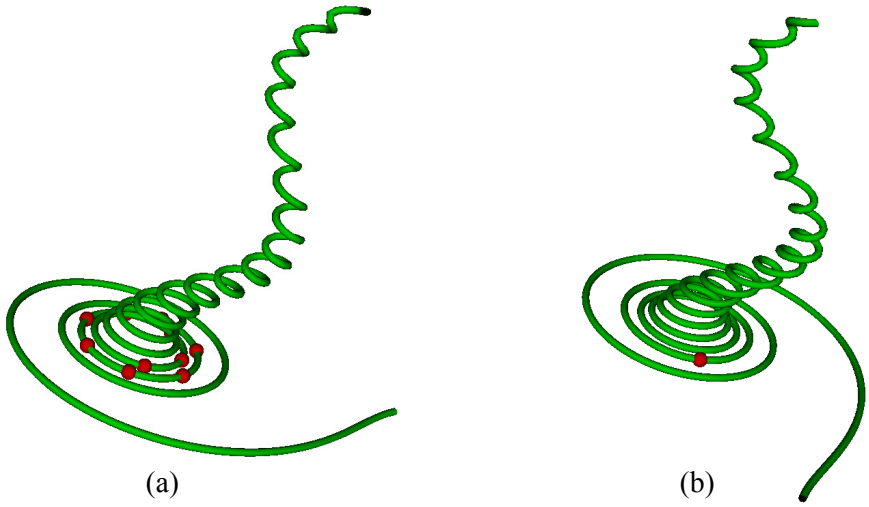


Figure 4.8: Remove redundant segmentation points: (a) nearby points (in red) are detected as segmentation points by our trained classifier. (b) only one segmentation point is left after post-processing.

Formally, assume that $S = \{s_i : 1 \leq i \leq N_S\}$ 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 N_S is the total number of segmentation points found by the classifier and $\alpha(s_i, s_j)$ measures the arclength between segmentation points s_i and s_j.

A group C of segmentation points is defined as:

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

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 (Section 4.3.1.3) 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 Equation 4.6. The point s_p (*resp.*, s_n) is an arbitrary point from the previous (*resp.*, next) group along the streamline. Since points in the same group are close to each other, picking an arbitrary point will hardly affect the final segmentation. Figure 4.8 (b) shows an example result of our grouping algorithm: only one point (the red point in (b)) is picked as the segmentation point from all its nearby segmentation points (the red points in (a)).

4.4 Results and Discussion

This section first discusses the results of applying the above method to a few data sets (Section 4.4.1). Next, Section 4.4.2 shows that the chosen features (Section 4.3.1) serve as good indicators for segmentation points, and especially the volume ratio of minimum bounding ellipsoids greatly improves the final results. Finally, Section 4.4.3 discusses how the parameters affect segmentation results.

The 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.

4.4.1 Flow feature extraction

To evaluate the performance of the above streamline segmentation algorithm, the resulting segments are clustered based on their similarities. The similarity between two segments is measured using the method introduced in Section 4.3.1.2, and then affinity propagation is applied to obtain the clusters. Three steady flow data sets (Table 4.1) 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, the streamlines used for training and the streamline segment clusters will be shown for each data set. The user-picked segmentation points are highlighted in purple, which are used as positive training examples. The remaining points in red are used as negative training examples. The segmentation was performed on the streamlines which were traced separately from the training streamlines

Table 4.1
The three flow data sets. The timing results are in seconds.

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

data set	segmentation		
	# lines	simplification time	segmentation time
tornado	150	1.99	35.76
five critical pts	150	0.60	2.74
solar plume	200	5.18	59.32

Case Study 1 – Tornado Data Set (Figure 4.9). Five (t_1-t_5) out of 30 traced streamlines were manually segmented. These five streamlines are the cluster representatives after clustering all the 30 streamlines. Note that users can specify a few nearby points as possible segmentation points (e.g., t_2). Streamline t_4 was not segmented because it does not contain any interesting 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, notice that a few streamlines from the cluster s_7 failed to be segmented as desired (circled in purple).

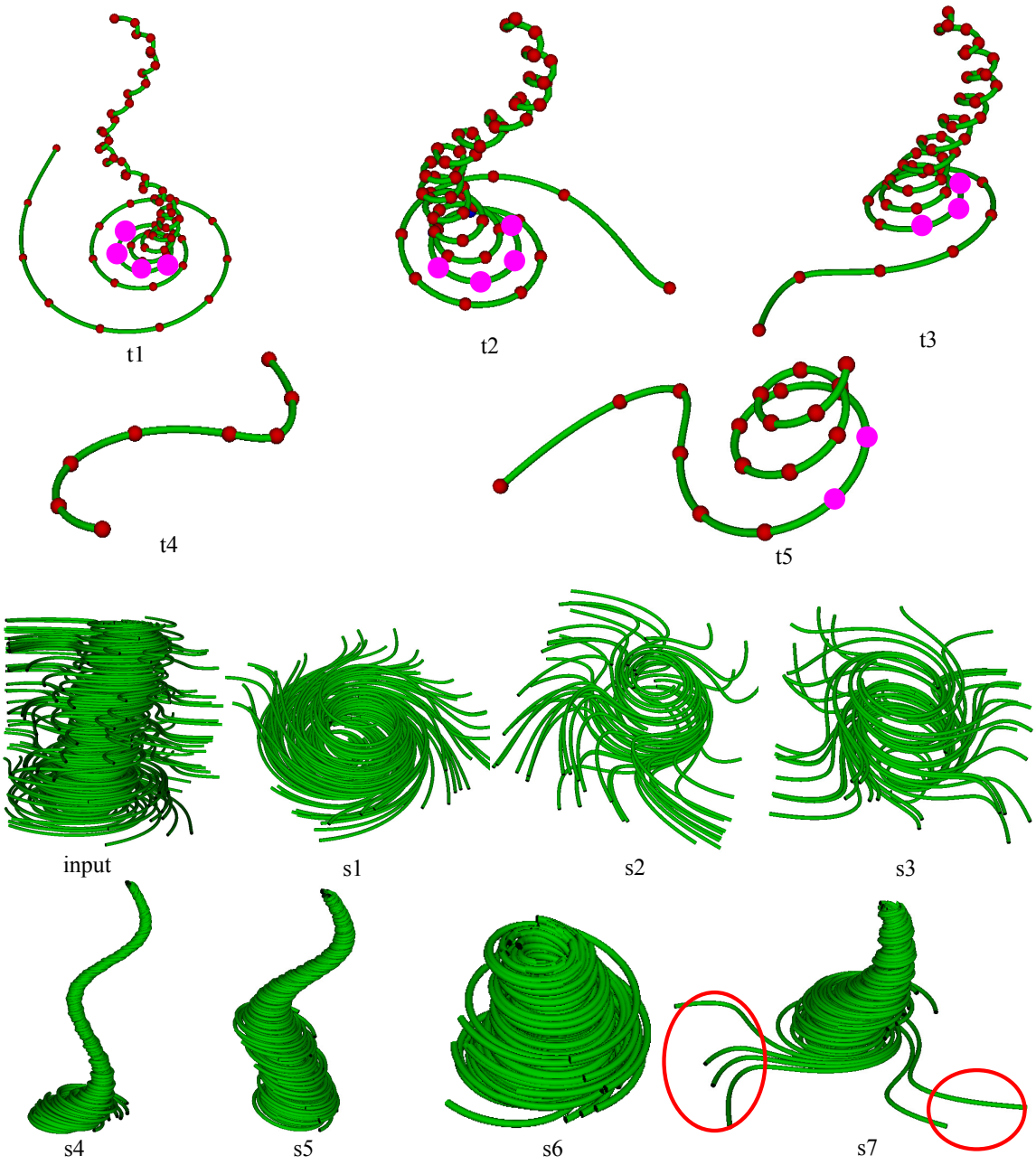


Figure 4.9: Five streamlines of the tornado data set were manually segmented (t_1-t_5) to train the classifier. Seven (s_1-s_7) clusters of streamline segments were generated.

Case Study 2 – Five Critical Points Data Set (Figure 4.10). 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 4.1) 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 4.11 and 4.12). Some streamlines from this data set are even more difficult to determine the "best" segmentation. So the 14 streamlines were segmented in a way such that interesting features were extracted. After segmentation and clustering, a total of 13 clusters are generated and shown in Figure 4.12. It can be seen from Figure 4.12 that the features specified during manual segmentation 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_{12} and s_{13}. Furthermore, the clusters s_1, s_2 and s_3 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 Figure 4.12, these spirals were successfully separated from the remaining streamlines, which suggests that the segmentation algorithm works well. Although segments in clusters s_7 to s_{11} are perceptually similar, they are unfortunately separated into different clusters because of the similarity measure and clustering algorithm. The same happened for clusters s_{12} and s_{13}. However, this is not the problem of the segmentation algorithm.

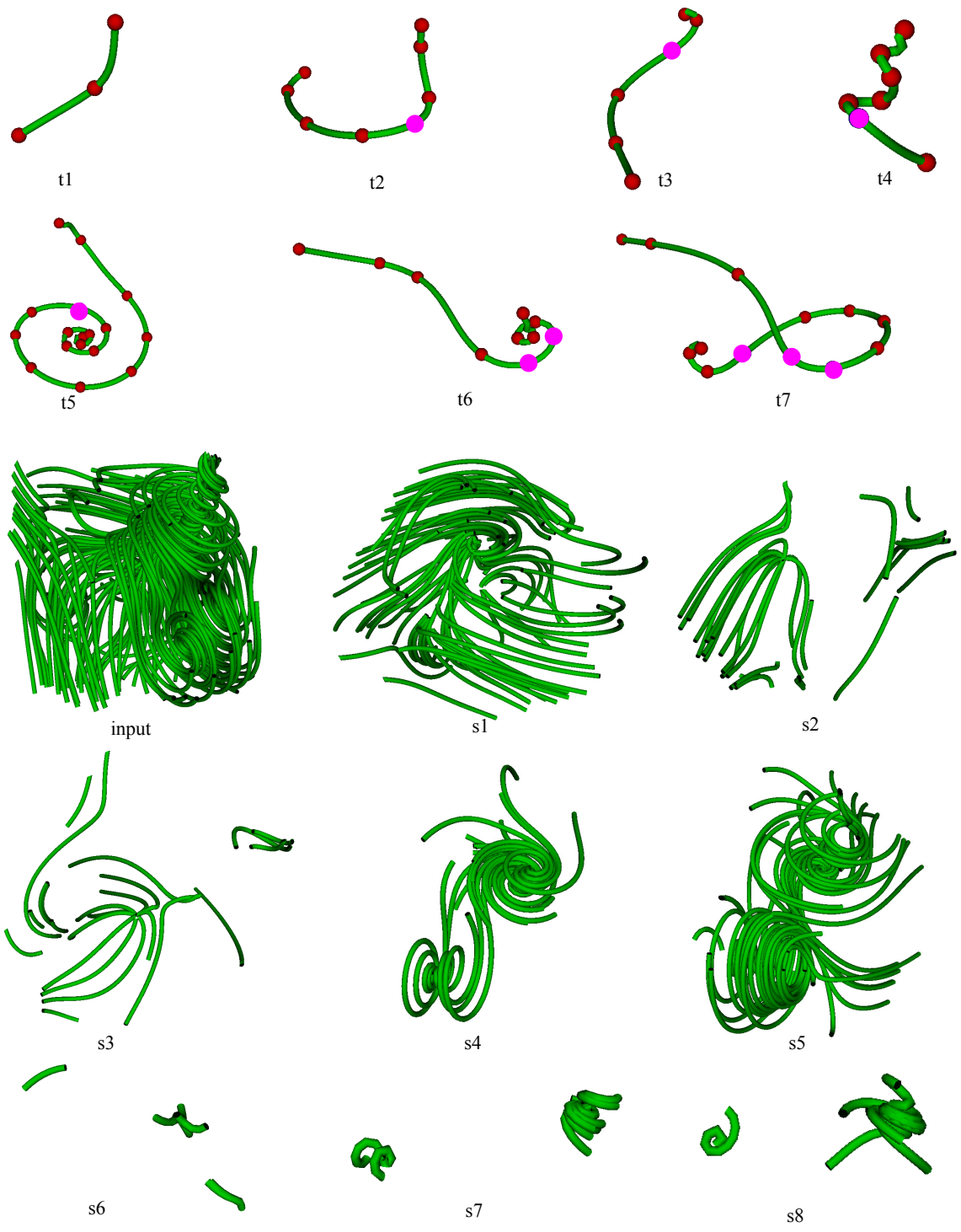


Figure 4.10: Seven streamlines of the five critical points data set were manually segmented (t_1-t_5) to train the classifier. Eight (s_1-s_8) clusters of streamline segments were generated.

Figure 4.11: Fourteen streamlines of the solar plume data set (t_1-t_8) were manually segmented for training.

Figure 4.11 (cont.)**:** Fourteen streamlines of the solar plume data set (t_9-t_{14}) were manually segmented for training.

Figure 4.12: Five (s_1-s_5) clusters of streamline segments of the solar plume data set are shown. Notice how the interesting features such as spirals and turbulent features are successfully extracted.

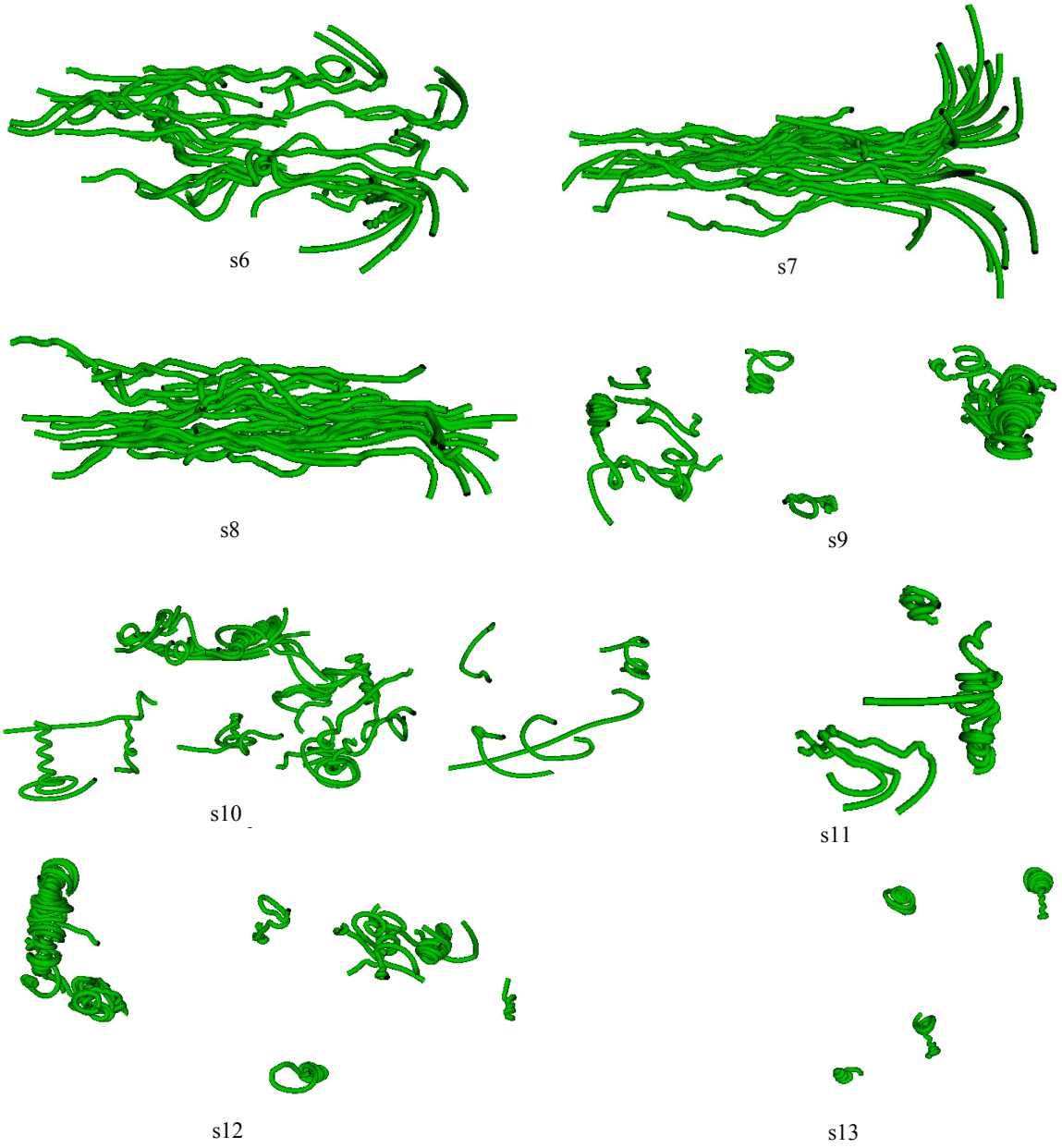


Figure 4.12 (cont.)**:** Streamline segment clusters s_6-s_{13} of the solar plume data set.

4.4.2 Feature selection

This section aims at showing 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 M_1, 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 $G_3 = \{M_4\}$. Comparing the segmentation results using G_1 and G_2 will show the contribution of M_4 because their only difference is whether M_4 is used, and comparing the segmentation results using G_1 and G_3 will show that using M_4 alone is not enough. For each data set, three different sets of training examples were generated using G_1, G_2, and G_3, and hence three classifiers were trained. Applying the three different classifiers to the same set of streamlines (without post-processing) will generate the segmentation results for comparison.

As Table 4.2 shows, classifiers trained using G_1 generally give the best segmentation results because it generates the least number of redundant segmentation points. A segmentation point is redundant if it appears in a region on a streamline where segmentation should not occur from a human's point of view. The redundant segmentation points are enclosed in black circles. In the first row of Table 4.2, a few

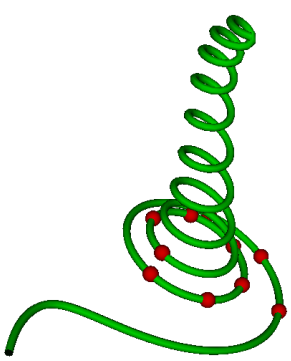
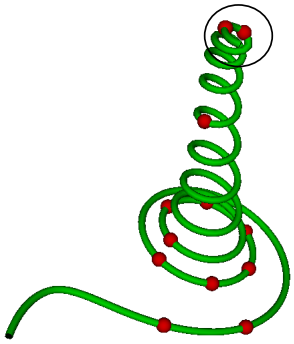
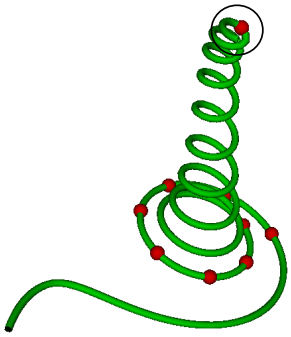
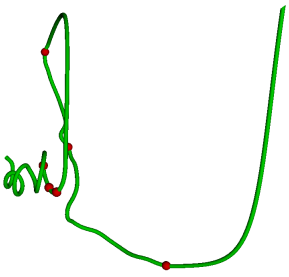
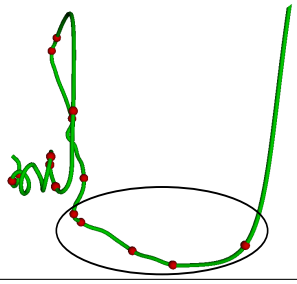
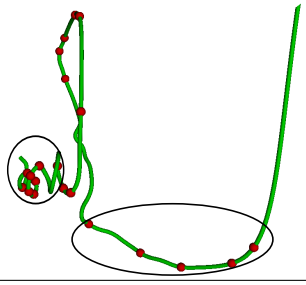
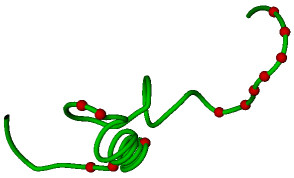
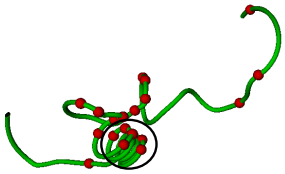
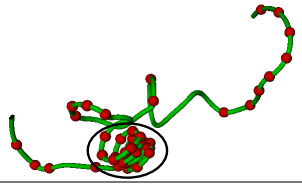
redundant segmentation points appear at the top of the streamline when G_2 and G_3 are used; in the second row, the redundant segmentation points appear in the middle; in the third row, the redundant points appear on the spiral. The table also shows that using G_1 requires the least amount of training time. By comparing columns G_1 and G_2, it can be seen that segmentation results are greatly improved when the volume ratio of minimum enclosing ellipsoids is incorporated into feature vectors. However, using it alone (G_3) does not give satisfactory segmentation results and also requires a much longer training time.

4.4.3 Parameters

There are two parameters that can affect the final segmentation results: (1) the number of different neighborhood sizes used during multiscale feature computation (Section 4.3.1), and (2) the distance threshold used to group nearby segmentation points (Section 4.3.4). Because the second parameter is computed based on the segmentation points already found by a classifier, users should not be allowed to adjust its value. The impact of the first parameter on the classifier performance in terms of AUC and training time will be discussed in the following.

The tornado data set is used in this experiment. In the beginning of the experiment, 36 positive and 298 negative examples are collected. A sequence of increasingly

Table 4.2
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 M_1, 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
		
		
		

data set	pos/neg examples	training time (in seconds)
tornado	28/195	G_1 : 6.59, G_2 : 8.23, G_3 : 30.69
crayfish	25/314	G_1: 20.22, G_2: 43.32, G_3: 259.44

larger neighborhood sizes is considered: 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40%, 45%, 50% of the total streamline points. The neighborhood sizes are increased by 5% because (1) at most 500 points are generated in tracing a streamline, and (2) a

streamline segment consisting of $500 \times 5\% = 25$ points is usually too tiny to be visually considered as a standalone segment. To determine the best range of neighborhood sizes to be used in multiscale feature vector computation, the following experiment is done: starting from each neighborhood size, n consecutive neighborhoods up to 50% of the total number of streamline points will be used. For example, if the smallest neighborhood size is 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 4.3 lists the statistics of AUC and training time for different neighborhood combinations.

Table 4.3 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 [21] 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. The segmentation results obtained by setting the starting neighborhood size to 5% and using 10 consecutive neighborhoods indeed were very good. However,

segmentation itself took a much longer time because more computation is required.

The values in Table 4.3 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.

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

4.5 Comparison

This section first compares the supervised streamline segmentation method with the other two segmentation algorithms [54, 89], and then compares the flow features extracted by the segmentation algorithm in this dissertation and that by Tao et al. [84].

Lu et al. [54] 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. Their algorithm is implemented using the EMD

Table 4.3
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	AUC/training time when 1-10 consecutive neighborhoods are used in multiscale feature computation									
	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									

distance between the two curvature and torsion histograms (Section 4.3.1.2) as the dissimilarity measure.

The major issue of their approach is that parameters t_s and t_l are not intuitive for users to adjust manually, which is illustrated by Figure 4.13 (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, the method in this dissertation 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. 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 [54].

The streamline segmentation algorithm by Wang et al. [89] is a bottom-up approach. A streamline is first split into *minimal segments* which are bounded by points of

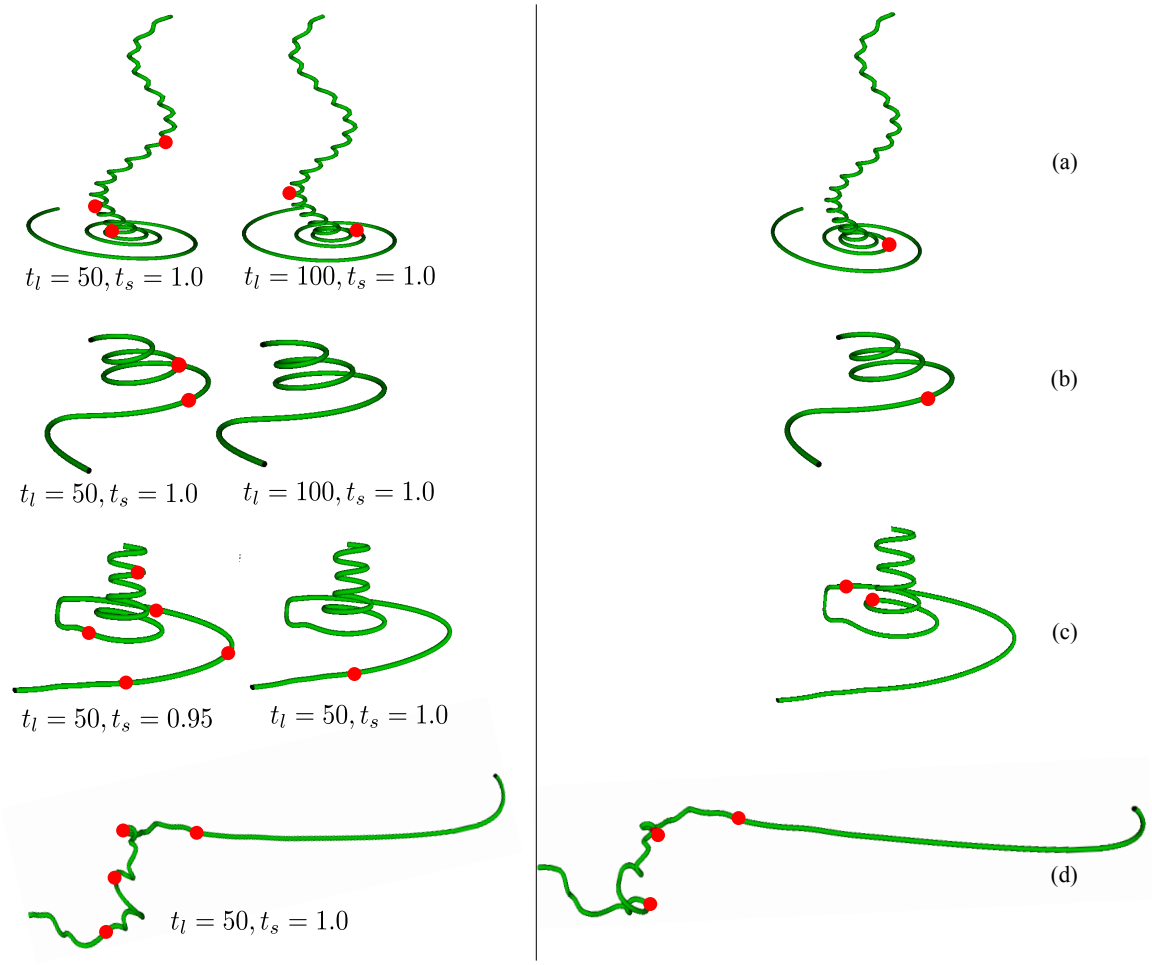


Figure 4.13: A comparison on streamline segmentation between [54] (left column) and the method in this dissertation (right column). The streamlines in row (a) and (b) are from the tornado data set, and those in rows (c) and (d) from the solar plume data set. The segmentation points are highlighted in red.

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, the binormal direction of a segment between two consecutive points is measured as the cross product of the velocity vectors at those points.

Wang et al. [89] 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, it is 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 [89]), but it is not a good choice to segment turbulent streamlines.

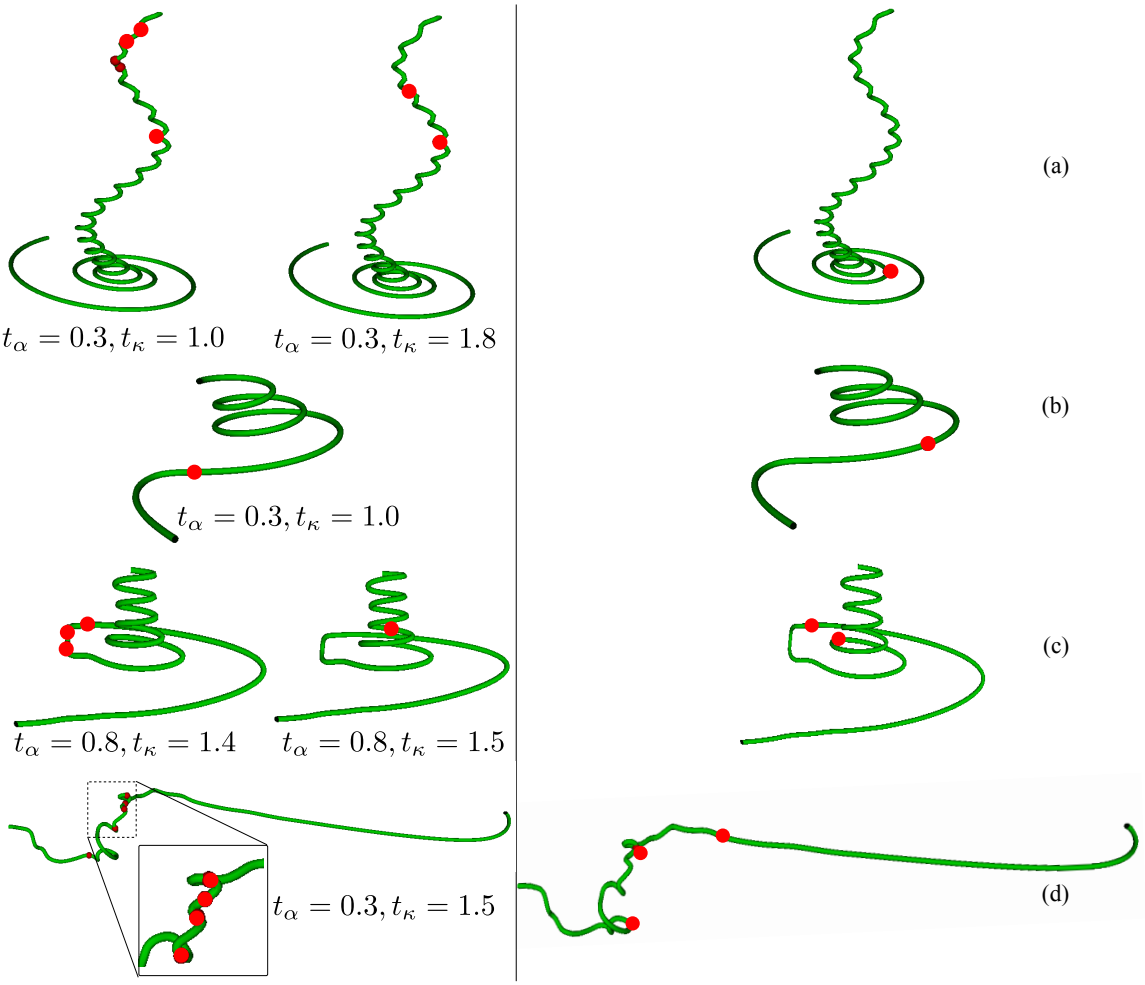


Figure 4.14: A comparison on streamline segmentation between [89] (left column) and the method in this dissertation (right column). The streamlines in row (a) and (b) are from the tornado data set, and those in rows (c) and (d) from the solar plume data set. The segmentation points are highlighted in red.

The remaining of this section compares the method in this dissertation with FlowString ([84]) on flow feature extraction. Tao et al. [84] 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 [1],

and is used in the following comparison.

Figures 4.15 and 4.16 show the features extracted by FlowString for tornado and solar plume 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 4.4.1). 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 4.15 (a)-(c), which look similar to each other. The features were extracted with minimum frequency and minimum length set to 100 and 3 respectively. For the tornado data set, the features found by the method in this dissertation were clustered into 7 distinguishable groups (Figure 4.9).

- Users need to adjust the value of minimum frequency (minimum length) to search for the desired features. For instance, the features in Figure 4.15 (e)-(f) were only available when the value of minimum frequency was decreased from 100 to 50 . For the solar plume data set, only two types of features (Figure 4.16 (a)-(b)) were extracted when minimum frequency and minimum length were set to 100 and 4 respectively. The spiral features in Figure 4.16 (d)-(f) did not appear unless the minimum frequency was set to a small value of 10. In contrast, since the method in this dissertation 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 4.15 (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 Figure 4.16 (c)-(f). However, the method in this dissertation is able to find more complete features (Figure 4.9 and 4.12).

The advantage of FlowString over the method in this dissertation 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, there is a limitation of the streamline segmentation method discussed in this dissertation: 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 [85], and incremental training should be performed.

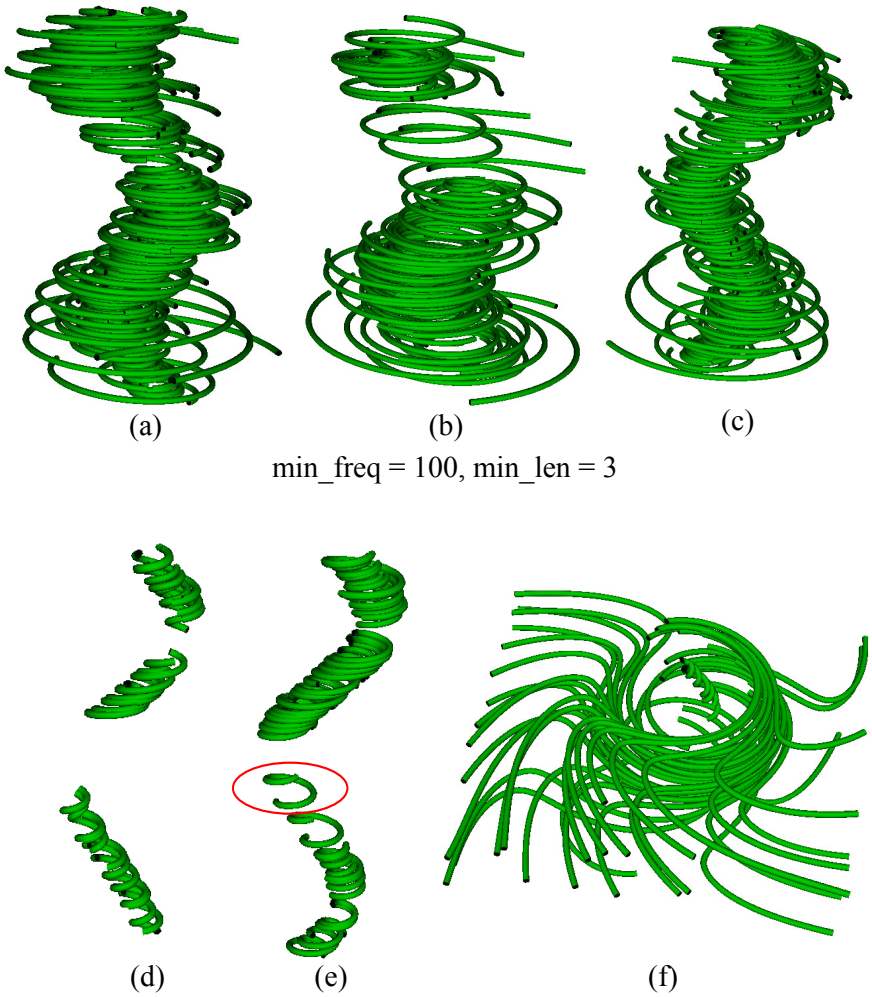
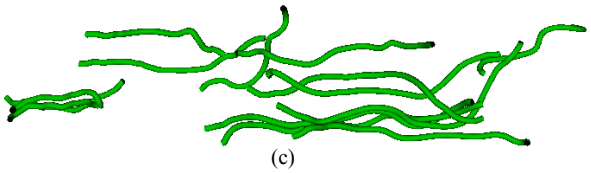


Figure 4.15: The features extracted by FlowString [84] for tornado data set.

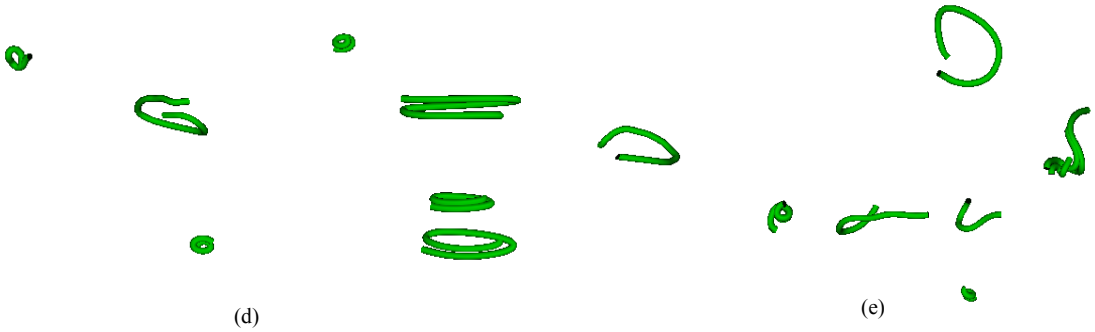
(a) (b)

min_freq = 100, min_len = 4



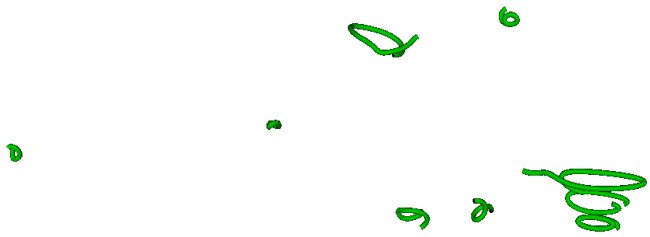
(c)

min_freq = 10, min_len = 3



(d) (e)

min_freq = 10, min_len = 3



(f)

min_freq = 10, min_len = 3

Figure 4.16: The features extracted by FlowString [84] for solar plume data set.

4.6 Conclusions

This chapter presents a novel streamline segmentation algorithm based on supervised machine learning. It is an early attempt of applying supervised machine learning to flow feature extraction. This chapter also discusses an effective heuristic for streamline segmentation: the volume ratio of minimum enclosing ellipsoids.

The supervised streamline segmentation algorithm first 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 redundant nearby segmentation points found by the classifier.

The results are encouraging, and the following items may be considered for future work:

- Identify more effective metrics besides the ones mentioned in Section 4.3.1 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 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 vector fields.

- Generate a hierarchy of coarse-to-fine segmentations for each streamline. Our current approach only generates a single segmentation. However, human tend to segment a geometric object in a hierarchical manner (e.g., hierarchical mesh segmentation [79]), where finer details are extracted deeper down in the hierarchy.

- 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 are available at `http://www.nd.edu/~cwang11/streamline-segmentation.html`

Chapter 5

DTEvisual: A Visualization System for Teaching Access Control using Domain Type Enforcement

The field of visualization includes many research areas, and flow field visualization is just one of them. Examples of other areas in visualization are information visualization [42] and education visualization [61]. DTEvisual is an education visualization tool developed for teaching access control in an elective senior-level operating system course.

This chapter first explains the motivation behind developing DTEvisual in Section 5.1. It then gives a detailed description of Domain Type Enforcement in Section 5.2. The architecture of DTEvisual and its main functions are presented in Section 5.3. It is followed by the evaluation of DTEvisual in Section 5.4. Future work to extend DTEvisual is discussed in Section 5.5. Finally, the conclusions are made in Section 5.6.

5.1 Motivation

Access control systems have become significantly more sophisticated in order to meet modern security requirements. An example is SELinux [5, 55, 56], a version of Linux developed by the National Security Agency. SELinux supports non-traditional models of access control, including Type Enforcement [13], Multilevel Security [8, 10], and Rolebased Access Control [74]. These modern systems are very complex. A strict Type Enforcement access control policy on a Linux system can contain tens of thousands of lines [55]. As another example, Windows attaches access control lists to objects. Windows access control lists now comprise up to 30 different privileges for operations on about 15 different kinds of objects [33]. It is important for educators to prepare students to deal with the complexity of these modern systems.

Domain Type Enforcement (DTE) [7] is an access control model widely used on Linux systems. Via DTE, a user essentially groups active entities (e.g., processes) into

domains; groups passive entities (e.g., files) into types; and specifies the kind of access each domain has to objects of a given type. A DTE policy is expressed in *Domain Type Enforcement Language* (DTEL). DTE has several characteristics that make it useful for education in access control [18]. First, DTE facilitates graphical expression of policies. Second, it allows representation of policies that conform to other access control models. DTE exposes students to a process-oriented paradigm for access control, drawing them into the application of the principle of least privilege naturally. It allows the policy designer to balance policy complexity against application of the principle of least privilege. Finally, it supports relatively concise expression of an access control policy via DTEL. DTEvisual was developed to make access control education using Domain Type Enforcement (DTE) more practical. It has been used within an elective senior-level operating system course. The feedback shows that this tool is very useful for classroom presentations, homework assignments, and self-study. This work has been published in the *Journal of Computing Sciences in Colleges* [47].

5.2 Domain Type Enforcement

Domain Type Enforcement views an operating system as a collection of processes and a collection of files. An attribute called *domain* is associated with each process, and another attribute called *type* is associated with each file. Domain is used to group a set of processes together which have the same access rights to certain domains and

certain types of files. DTE Language (DTEL) is a high-level symbolic language for expressing DTE specifications in a human-friendly form. DTEL provides four primary statements for expressing a DTE specification: the *type* statement, the *domain* statement, the *initial_domain* statement, and the *assign* statement.

A DTEL `type` statement declares one or more types to be part of a DTE specification; other DTEL statements may refer only to types declared with the type statement. For example, the following type statement declares two types `readable_t` and `writable_t`:

```
type readable_t, writable_t;
```

A DTEL `domain` statement defines three components:

1. entry point programs, identified by the path name, that are bound to the domain and must be invoked in order to enter the domain.

2. permissible modes of access, when executing in the domain, to the files of specified types. There are five possible modes of access: 'c', 'r', 'w', 'x' and 'd'. Mode 'c' is for creating files. Modes 'rwx' are borrowed from the normal UNIX modes. UNIX uses 'x' mode for directory traversal; DTEL distinguishes between execute and traverse access using a new mode, 'd', that applies only to directories.

3. permissible modes of access, when executing in the domain, to the processes in other specified domains. DTEL provides two access rights for creating new processes. If a domain A has *exec* access rights to another domain B, a process in A may create a process in B by executing one of B's entry point programs and requesting that the program run in B. If a domain A has *auto* access rights to another domain B, a process in A automatically creates a process in B when it does a normal *exec*() system call of an entry point program of B.

Each domain statement is a list of tuples where each tuple either contains a UNIX path or a collection of access modes (designated by "`->`") to a collection of type or domain names. Following is an example of domain statements. To enter the domain `daemon_d`, the program `/sbin/init` needs to be executed. The processes in this domain can read and traverse all the files/directories whose type are either `generic_t`, `readable_t`, or `writable_t`, and they can create and write the files with a type `writable_t`. The processes in `daemon_d` are allowed to create a process in `login_d` when they execute `login_d`'s entry point program:

```
domain daemon_d = (/sbin/init),
                  (dr->generic_t,readable_t),
                  (cdrw->writable_t),
                  (auto->login_d);
```

The `initial_domain` statement specifies the domain of the first process executed by an operating system. In the following example, the first process will run in domain `daemon_d`:

```
initial_domain = daemon_d;
```

The `assign` statement is used to associate a type with a path P and is optionally recursive; recursive statements apply to all paths having P as a prefix. If a file is renamed, its assignment statement is changed to reflect the file's new location. DTEL provides a feature to force type assignments to be static which locks the type of a file to be the same as what is specified in the DTE specification. Each assignment statement starts with the keyword `assign`. The keyword is then followed by either a flag "-r" indicating recursive assignment, or a flag "-s" indicating static assignment, or both of them. Following is an example of assignment statements. All the files under `/etc` and its subdirectories are assigned a type `readable_t`. All the files under `/dte` and its subdirectories are given a type `dte_t`, and their type will not be changed.

```
assign -r readable_t /etc;
assign -r -s dte_t /dte;
```

5.3 System Overview

DTEvisual aims at making the daunting task of writing complex DTE specification much easier for beginners. The visualization and interaction features of this system will facilitate the learning process. The main features of DTEvisual are the following:

- Users can design and edit a specification visually and then save the resulting policy.
- DTEvisual allows users to display a specification using two types of graphs: the *general graph* and the *type graph* (Section 5.3.2).
- Users can use DTEvisual to carry out queries about a specific policy.

The user loads a DTE specification given in either DTEL (*.dte) or visualization descriptions (*.dtevis). When a DTE specification in DTEL is loaded, DTEvisual uses a graph layout algorithm [27] to determine the positions of domain and type nodes in the corresponding general and type graphs. The user can visually modify the generated graphs and save the results to a dtevis file which records both the DTE specification and the layout of the graphs. Next time, the same DTE specification can be loaded through the dtevis file without worrying about how the general and type graphs should be displayed.

5.3.1 User Interface

Figure 5.1 shows the main user interface of DTEvisual, which contains the menus, toolbar, and visualization area. DTEvisual provides separate File, Edit, View, Practice, Settings and Help menus, as well as a toolbar. The File, Edit and View menus are used to access functionality related to loading/saving, editing and viewing specifications. The Settings menu is for enabling the query animation function and setting animation intervals (Section 5.3.4). The Practice menu allows students to access a test for evaluating student learning.

Commonly used operations are classified into the following five groups in the toolbar (Figure 5.2):

1. Group A has I/O-related icons (e.g., importing and exporting a file).

2. Group B has buttons for changing editing modes in a general graph (Section 5.3.2). The buttons from left to right are for: (1) moving nodes in a graph, (2) highlighting part of a graph, (3) adding domain nodes, (4) adding type nodes, (5) adding auto access between two domain nodes, (6) adding exec access between two domain nodes, and (7) adding permissible accesses from a domain node to a type node.

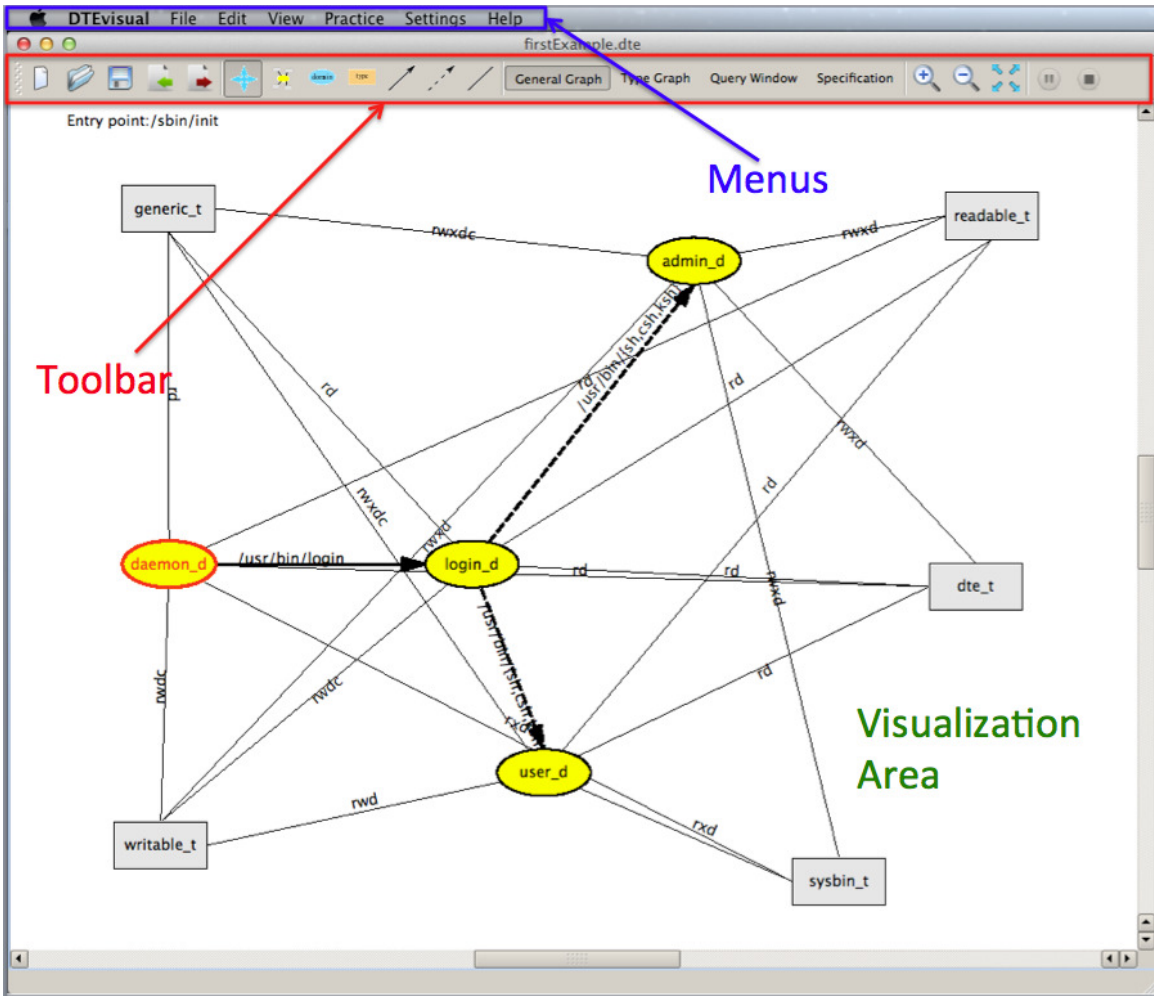


Figure 5.1: Main User Interface

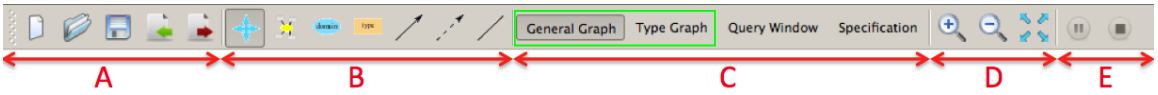


Figure 5.2: DTEvisual System Toolbar

3. Group C contains shortcuts to different functions for policy analysis (e.g., opening the query window).

4. Group D contains icons for the user to zoom in and out and enter/exit the full-screen.

5. Group E has two icons for controlling query animation.

5.3.2 Domain and Type Graphs

DTEvisual provides two modes of visualization of a DTE specification: the general graph and the type graph. The user can switch between these two graphs by clicking the buttons (in green box) in Group C in Figure 5.2.

The general graph depicts domains, types, transitions between domains, and access of domains to a given type. In Figure 5.3, ellipses and rectangles denote domain nodes and type nodes, respectively. Thick solid and dashed directed edges between domains are 'auto' and 'exec' transitions, respectively, and edge labels are entry points. Domain and type nodes are connected with undirected edges, and each edge label indicates the permission of a domain has on a type.

The type graph shows file types. In Figure 5.4, it is displayed with a radial tree [86]. Directories and files at the same level are placed on a dotted circle with static types shown in double circles, and a trailing slash is used to differentiate between files and directories.

Since a general graph may become cluttered when the number of nodes increases, DTEvisual allows the user to choose whether to display the edges between domain

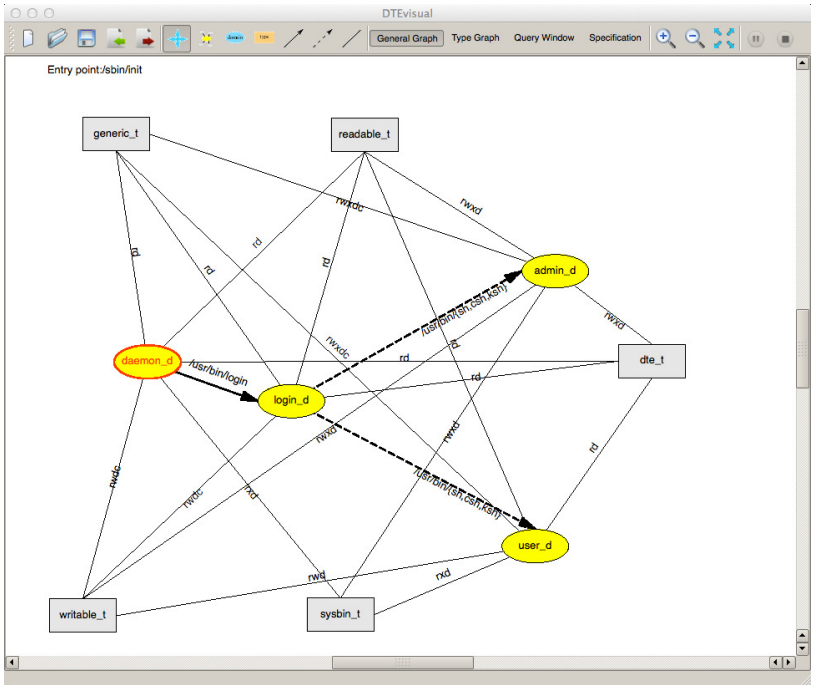


Figure 5.3: Domain graphs

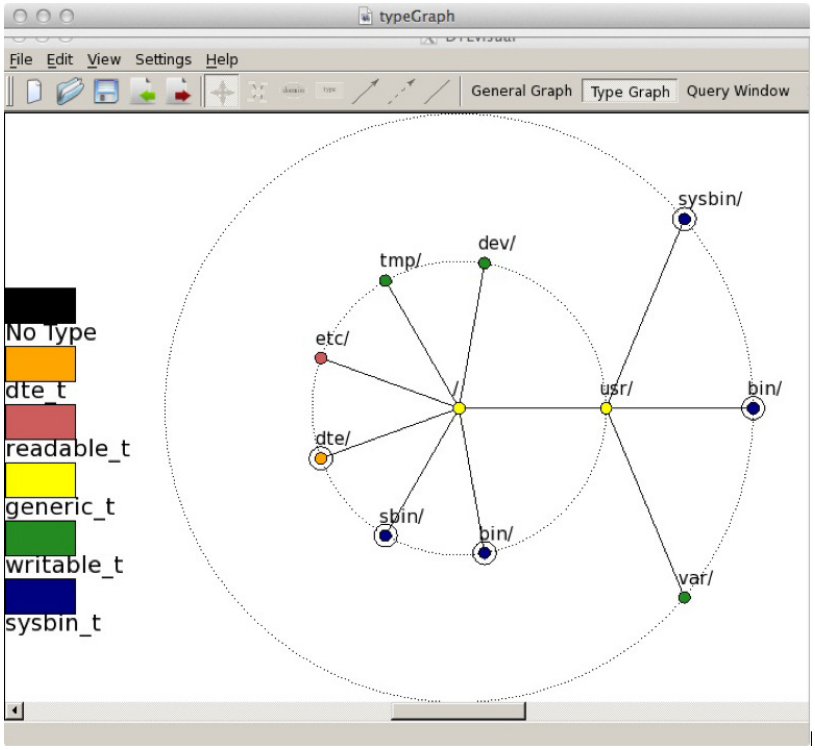


Figure 5.4: Type graphs

nodes or between domain and type nodes, as illustrated by the View menu in Figure 5.5.

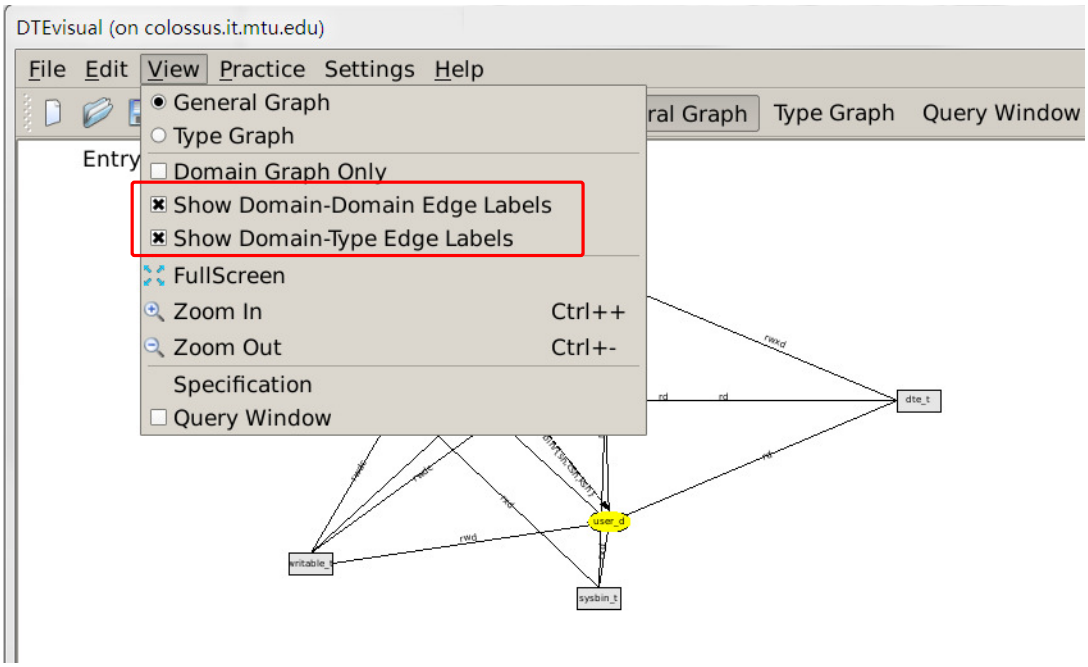


Figure 5.5: Toggle the display of edges in the general graph

Another way of reducing visual clutter is to allow the user to highlight part of a general graph. Figures 5.6 to 5.8 demonstrate this feature. Figure 5.6 shows that the first time the user clicks a type node '`readable_t`', the selected item and its adjacent edges and nodes are highlighted while the rest is grayed out. Figure 5.7 shows that the second click on the same item does the opposite: only the rest of the general graph is highlighted. Finally, Figure 5.8 shows that the third click brings the rendering back to normal mode.

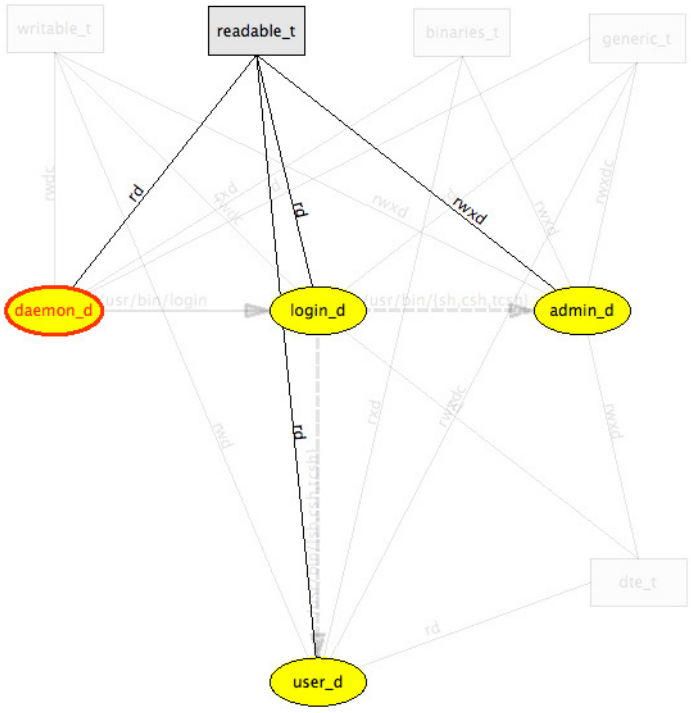


Figure 5.6: First click on the type node '`readable_t`' highlights the node and its adjacent edges and nodes.

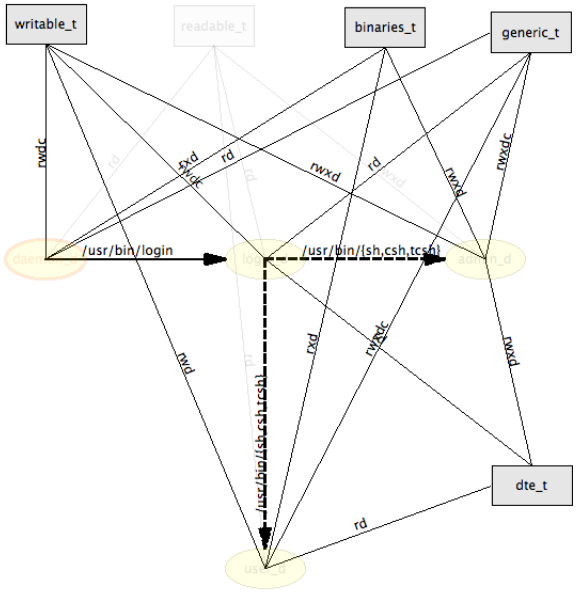


Figure 5.7: Second click on the type node '`readable_t`' highlights the part not highlighted in Figure 5.6.

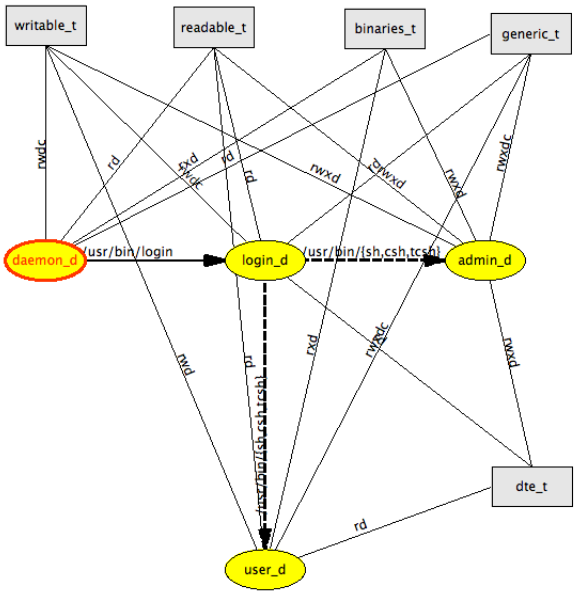


Figure 5.8: Third click on the type node '`readable_t`' brings the rendering back to normal.

5.3.3 Graph Editing

The user can add domain nodes, type nodes, or edges between them in a general graph. A domain (or type) node is added by clicking at the desired position and a nonempty but unique name is required for the new node. Dragging from a node to the other adds an edge; however, no edge will be added if the user attempts to add an invalid type or duplicate an edge. An appropriate label must be added to each newly created edge. For example, the label of a domain-type edge should indicate valid permissions that only contain characters 'c', 'r', 'x', 'w' and 'd'. The context menu of a node/edge permits the user to delete or change the name of that node/edge. Once a node is deleted, all adjacent edges are also deleted.

DTEvisual allows the user to modify a type assignment statement and the modifications will be immediately reflected in the type graph. For a type graph, the user may change the name, type or flag of a node, delete a type, and assign the type to a child node. All operations are accessed from a nodes context menu (Figure 5.9). DTEvisual performs various checks to ensure no semantic errors are introduced when the user edits a type graph. When the type of a node is changed, DTEvisual checks each of its children to ensure if there is any child which inherits the nodes type, and those who are affected will receive the new type. The user may create a new type in the process of changing types, and a node representing the new type will be automatically added

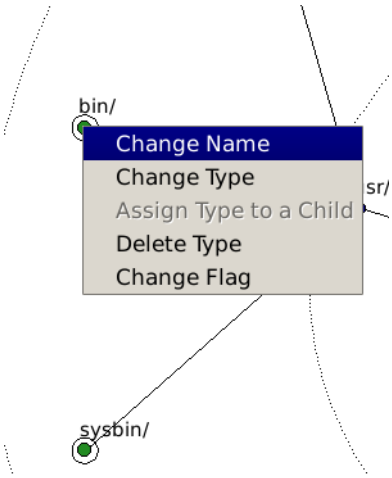


Figure 5.9: Context Menu of Type Node

to the general graph. A node inherits its parents type when its type is deleted. If the inherited type is different from its original one, it is treated as if the nodes type has been changed. The ability of adding the '-s' flag (Section 5.2) to a node is disabled if there exists a node among its children which has a different type, and the ability of removing '-r' flag (Section 5.2) is disabled for those nodes with children (i.e., directories) because directories differ from files by the existence of the '-r' flag. The same principle is used when the user adds a child to a node: the child and its parent have the same type if its parent has the -s flag set on, and no child can be added to any node without the -r flag.

5.3.4 Queries

DTEvisual can perform the following queries:

1. What is the type of a file?

2. What files can be accessed by a process?

3. What files can be accessed by a process in a certain mode?

4. From which domains a file can be executed?

5. What domains can a file access with a specified permission?

6. Are there files without any type assigned?

7. Are there files which cannot be accessed with a certain permission?

In order to run a query, the user must first bring up the query window (Figure 5.10) either by right-clicking in the empty visualization area or referring to View menu. After all the required inputs for a query have been given, the query is executed by hitting the Run button and the results appear in the Query Output.

DTEvisual is able to show a real-time animation of how the answer to a query is found. The search for the answer to a query usually can be broken into a few individual steps. Information related to each of these steps is visualized during the animation. For example, the animation of the query "From which domains the file '`/usr/bin/lp`' can be executed" is shown in Figures 5.11 and 5.12 because it takes two steps to answer this query. In the first step (Figure 5.11), the type of '`/usr/bin/lp`' is found with the corresponding type node '`binaries_t`' highlighted in red. In the second step

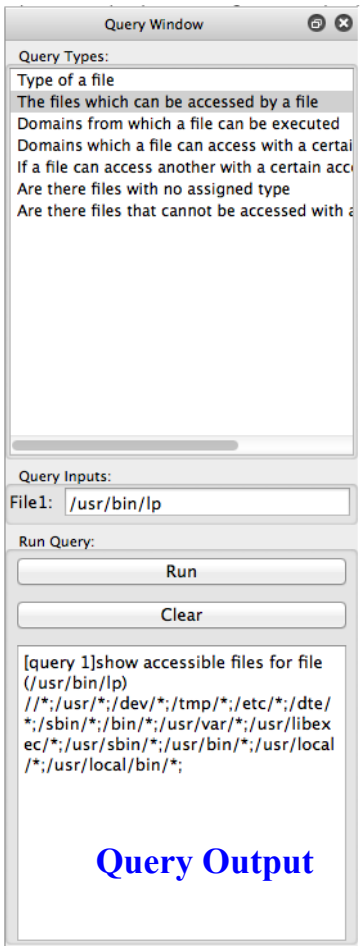


Figure 5.10: DTEvisual Query Window

(Figure 5.12), all the domains which have executable permissions on '`binaries_t`' are highlighted in green, and the permissible access modes from these domains to the type node are shown in blue. The user can enable/disable or change the speed of this real-time animation in the Settings menu. During the animation, the user can pause/resume the animation. It is also possible to re-run a query quickly by double-clicking the query line (a line which contains "[query x]") in the query output box (Figure 5.10).

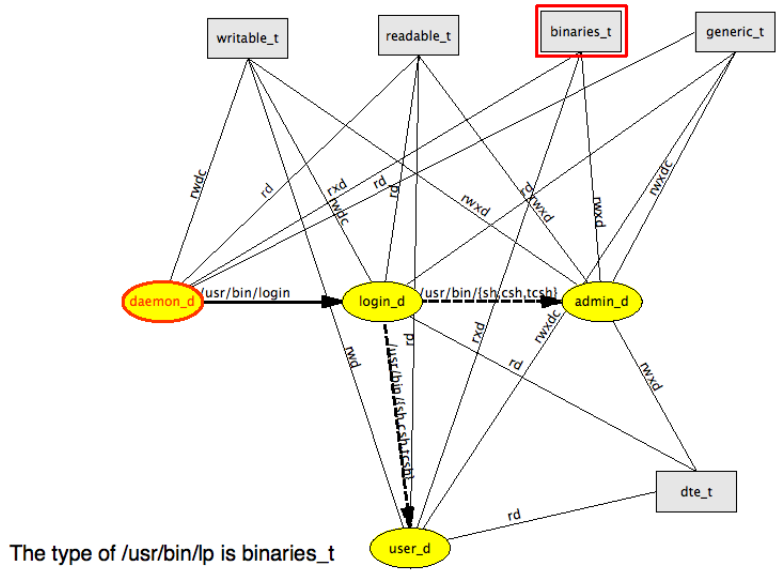


Figure 5.11: Determine the type of '`/usr/bin/lp`', which is '`binaries_t`'

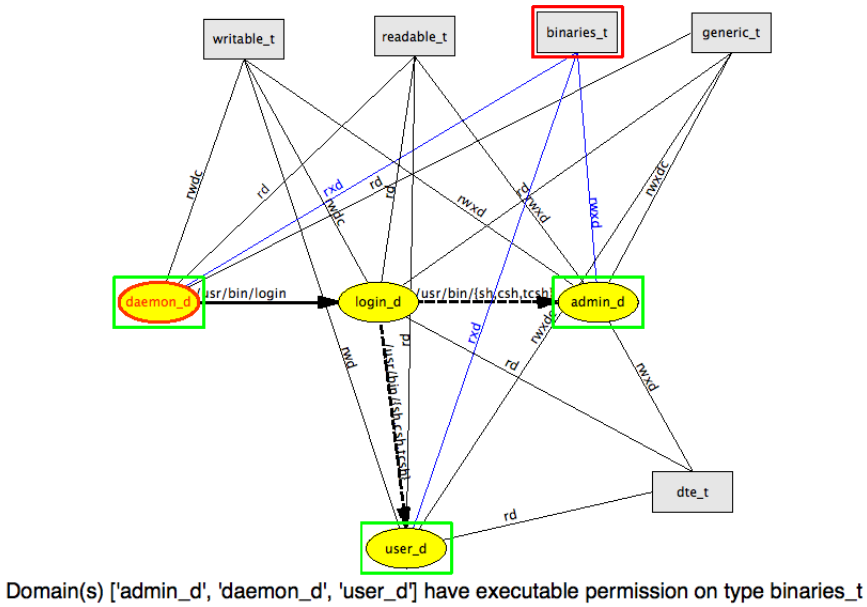


Figure 5.12: Find the domains that have executable permissions (x) on '`binaries_t`'

5.4 Evaluation

DTEvisual was used in a senior-level operating systems elective CS4411. Students in this course have already taken courses in systems programming and concurrent computing. The course had an enrollment of ten. This course covers scheduling, deadlock, memory management and virtual memory, file systems, and operating system security. Topics in access control included the Bell-LaPadula model, DTE, and Role-based access control (in this order). The Bell-LaPadula model [8] is often used for enforcing access control in government and military applications. It describes a set of access control rules which use security labels on files and clearances on users. With Role-based access control (RBAC) [74], users are assigned particular roles, and through those role assignments acquire the computer permissions to perform particular functions.

DTEvisual was used to present examples of DTE policies graphically. This was especially useful for modifying policies during classroom discussions. Students also used the system to complete homework problems. With a tool that simplifies presentation, the professor was able to leverage DTE for several lessons in access control. DTEvisual was used to present a DTE example policy that restricts user processes from writing to the system binaries. This motivated a discussion on trade-offs between policy complexity and application of the principle of least privilege. The initial example was

modified to confine a specific binary to a specific set of files. This further illustrated operation of a DTE specification. This example was also used to demonstrate the inadequacy of UNIX discretionary controls to achieve this same confinement. The expression of a Bell-LaPadula model was then explored using DTE. In addition to other problems, students were given an assignment to depict (with paper and pencil) an RBAC policy that was expressed in DTEL, and to modify that DTEL policy to extend the given RBAC policy. This helped them to understand the use of a hierarchy for expressing permissions (since there is no inheritance in DTE) and, at a higher level, how different models are suited to different access control problems.

At the end of the course, a survey was conducted to get the students' feedback on DTEvisual. Since the class had only 10 students, a very small sample, no advanced statistical analysis was attempted. The comments received were uniformly positive about (1) the usefulness of the general graph and type graph for understanding a policy (mean 4.5 and standard deviation 0.58), (2) the utility of the tool for finding mistakes in a DTE policy (mean 4.0 and standard deviation 0.82), (3) the use of DTE to increase understanding of the balance between the principle of least privilege and policy complexity (mean 4.5 and standard deviation 0.58), and (4) the use of DTE to better understand the RBAC and Bell-LaPadula models (average 4.0 and standard deviation 0.82). Students also gave high marks to representation and layout of various graphs, and the uses of font sizes, labels, and colors with averages no less than 4.25 and standard deviations between 0.5 and 0.58.

5.5 Future Work

The experience shows that DTEvisual has the potential to be developed into a full blown visualization and pedagogical environment for learning and teaching access control. DTEvisual may be extended in several ways. First, DTEvisual is a quick prototype implemented in Python with some tools such as Qt. As a result, its efficiency can be improved by using a programming language such as C++. Second, to support large and realistic access control systems, new features must be added to address issues created by the large number of nodes and edges in the general and type graphs. The plan is to use a Focus+Context technique so that the user is able to see the objects of primary interest in full detail while maintaining an overview of all surrounding information. Third, the query system can be redesigned to have an open structure so that an instructor is able to add new and remove existing queries. Fourth, DTEvisual can be extended to become a programming+visualization environment [19]. The class libraries can be used to intercept requests to the access control system in order to indicate what would be the response on a system that applied the DTE policy. These class libraries will at the same time provide information to the visualization system. In this way, the user is not only able to visualize DTE related activities in real time and receive feedback from the visualization system, but also use the system to debug code and policies.

5.6 Conclusions

DTEvisual provides a system that makes teaching access control using DTE more practical. It facilitates graphical depiction, construction, and modification of a DTEL policy. It has an interactive graphical exploration mode that can isolate selected portions of a depicted policy. Finally, a query subsystem allows precise determination of the set of files that can be accessed by a specified process. This tool has been useful for classroom presentations, homework assignments, and self-study. Better education on modern access control systems should encourage more widespread adoption of sophisticated approaches and have a significant impact on system security for the systems ultimately managed by these students. The system and example policies are available for download from `http://www.cs.mtu.edu/~jmayo/dtevisual/`.

Chapter 6

Conclusions

This dissertation focuses on the topic of feature extraction in flow visualization. Extracting flow features provide a viable solution for reducing visual clutter an occlusion in a vector field. In summary, this dissertation makes the following contributions:

- Proposes a novel method of measuring streamline similarity inspired by bag-of-features, and shows the utility of this method by applying it to streamline query and clustering. The proposed similarity measure between two streamlines is independent of their relative positions and orientations, and takes into account the distances among features on each streamline. It generally performs better at finding out similar streamlines than some existing methods; however, it is not perfect as illustrated in Section 3.5.3 since some streamlines which look

similar from a human perspective may appear in different clusters. Therefore, the proposed method may be improved in the following ways: (1) searching for features to more effectively describe the shape of a streamline, and (2) leveraging supervised machine learning to obtain a better distance measure.

- Applies supervised machine learning to the problem of streamline segmentation. The purpose of streamline segmentation is to separate distinct features along a streamline. Compared with some of the existing streamline segmentation methods which require users to adjust a few parameters for segmentation, the proposed one learns a non-trivial decision function of different parameters, and usually performs better at separating different features along a streamline. This feature-based streamline segmentation is fundamental to many flow data analysis tasks such as feature detection and pattern querying. It is also general enough to be used in other fields such as computer vision, where curve segmentation is used to extract features of 2D curves.

- Proposes an effective heuristic based on minimum volume bounding ellipsoids which can be used to help determine whether segmentation on a streamline should be taken or not. This heuristic improves streamline segmentation results. Other applications which require a streamline similarity measure may also benefit from this heuristic. One disadvantage of this approach is its relatively high computation cost due to the iterative optimization procedure for finding out the minimum volume bounding ellipsoid of a streamline segment.

- Presents a software tool DTEvisual which can be used to make teaching access control more effective. The tool has been useful for classroom presentation, homework assignments, and self study. Currently, DTEvisual is not suitable for visualizing very complicated DTE polices because the visualization will become too cluttered to be useful for users.

In the future, it is interesting to see how machine learning can be leveraged to extend the above works in flow visualization. First, learning to rank may be used to improve the streamline similarity measure. The goal is to obtain a ranking function which can decide the level of similarity between two streamlines based on training examples. This type of ranking function has been widely used in information retrieval tasks such as web search: given a query, the list of documents ranked in decreasing order of relevance is returned. Second, experimenting different machine learning algorithms for a given problem is a common practice to determine which algorithm suits the problem best. Since only SVM has been tested for streamline segmentation, it is interesting to check how other supervised machine learning algorithms perform for this problem.

For DTEVisual, it is worthwhile to leverage existing techniques for the visualization of large graphs to visualize a real world DTE policy. Such a DTE policy may be very complicated and hence requires a large number nodes and edges in a general/type graph to be visualized. Existing large graph visualization techniques allow users to

see the objects of primary interest in full detail while maintaining an overview of all surrounding information. With such capability, DTEvisual may become a useful tool for system administrators instead of just being a pedagogical tool. Efficiency is currently an issue for DTEvisual because it was developed in Python. In order to support visualizing large graphs, some critical parts of the system may need to be written in C++.

References

[1] FlowString: Partial streamline matching. `http://www3.nd.edu/~cwang11/flowstring.html`. Accessed: 2015-3-28.

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

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

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

[5] N. S. Agency. Security enhanced linux. `https://www.nsa.gov/research/selinux/index.shtml`. Accessed: 2015-03-30.

[6] L. Augsburger, P. Reymond, E. Fonck, Z. Kulcsar, M. Farhat, M. Ohta, N. Stergiopulos, and D. Rüfenacht. Methodologies to assess blood flow in cerebral

aneurysms: Current state of research and perspectives. *Journal of Neuroradiology*, 36(5):270–277, 2009.

[7] L. Badger, D. F. Sterne, D. L. Sherman, K. M. Walker, and S. A. Haghighat. Practical domain and type enforcement for unix. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, pages 66–77, 1995.

[8] D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations. Technical Report MTR-2547, MITRE Corporation, Bedford, MA, 1973.

[9] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD Workshop*, pages 359–370, 1994.

[10] K. J. Biba. Integrity considerations for secure computer systems. Technical Report MTR-3153, MITRE Corporation, Bedford, MA, 1977.

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

[12] D. E. Blair and 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):209–214, 2011.

[13] W. E. Boebert and R. Y. Kain. A practical alternative to hierarchical integrity policies. In *Proceedings of the 8th National Computer Security Conference*, volume 18, 1985.

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

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

[16] A. M. Bronstein, M. M. Bronstein, L. J. Guibas, and M. Ovsjanikov. Shape google: Geometric words and expressions for invariant shape retrieval. *ACM Transactions on Graphics*, 30(1):1, 2011.

[17] R. Bujack, I. Hotz, G. Scheuermann, and E. Hitzer. Moment invariants for 2D flow fields using normalization. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 41–48, 2014.

[18] S. Carr and J. Mayo. Teaching access control with domain type enforcement. *Journal of Computing Sciences in Colleges*, 27(1):74–80, 2011.

[19] S. Carr, J. Mayo, and C.-K. Shene. Threadmentor: a pedagogical tool for multithreaded programming. *Journal on Educational Resources in Computing*, 3(1):1, 2003.

[20] J. R. Cebral, M. A. Castro, J. E. Burgess, R. S. Pergolizzi, M. J. Sheridan, and C. M. Putman. Characterization of cerebral aneurysms for assessing risk of rupture by using patient-specific computational hemodynamics models. *American Journal of Neuroradiology*, 26(10):2550–2559, 2005.

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

[22] W. Chen, S. Zhang, S. Correia, and D. S. Ebert. Abstractive representation and exploration of hierarchically clustered diffusion tensor fiber tracts. In *Computer Graphics Forum*, pages 1071–1078, 2008.

[23] 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.

[24] S. Cohen and L. Guibas. The earth mover's distance under transformation sets. In *Proceedings of IEEE International Conference on Computer Vision*, pages 1076–1083, 1999.

[25] I. Corouge, S. Gouttard, and G. Gerig. Towards a shape model of white matter fiber bundles using diffusion tensor mri. In *IEEE International Symposium on Biomedical Imaging: Nano to Macro*, pages 344–347, 2004.

[26] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, 2012.

[27] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics*, 15(4):301–331, 1996.

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

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

[30] Y. Freund, R. Schapire, and N. Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[31] B. J. Frey. Affinity propagation faq. `http://www.psi.toronto.edu/affinitypropagation/faq.html`. Accessed: 2013-07-19.

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

[33] S. Govindavajhala and A. W. Appel. Windows access control demystified. Technical Report TR-744-06, Princeton university, Princeton, NJ, 2006.

[34] D. Guo. *Automated feature extraction in oceanographic visualization*. PhD thesis, Massachusetts Institute of Technology, 2004.

[35] G. Hall and J. M. Watt. *Modern numerical methods for ordinary differential equations*. Oxford University Press, 1976.

[36] F. Hitchock. The distribution of a product from several sources to numerous locations. *Journal of Mathematical Physics*, 20:224–30, 1941.

[37] M. Hollander, D. A. Wolfe, and E. Chicken. *Nonparametric statistical methods*. John Wiley & Sons, 2013.

[38] M. Isenburg and J. Snoeyink. Face fixer: Compressing polygon meshes with properties. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 263–270, 2000.

[39] R. Jianu, C. 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.

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

[41] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.

[42] D. Keim et al. Information visualization and visual data mining. *Visualization and Computer Graphics, IEEE Transactions on*, 8(1):1–8, 2002.

[43] R. Kimmel, C. Zhang, A. M. Bronstein, and M. M. Bronstein. Are MSER features really interesting? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(11):2316–2320, 2011.

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

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

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

[47] Y. Li, S. Carr, J. Mayo, C.-K. Shene, and C. Wang. Dtevisual: a visualization system for teaching access control using domain type enforcement. *Journal of Computing Sciences in Colleges*, 28(1):125–132, 2012.

[48] Y. Li, C. Wang, and C.-K. Shene. Streamline similarity analysis using bag-of-features. In *Proceedings of IS&T/SPIE Conference on Visualization and Data Analysis*, 2014.

[49] Y. Li, C. Wang, and C.-K. Shene. Extracting flow features via supervised streamline segmentation. *Computers & Graphics*, 52:79–92, 2015.

[50] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[51] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[52] J. L. Lowther and C.-K. Shene. Rendering+ modeling+ animation+ postprocessing= computer graphics. *Journal of Computing Sciences in Colleges*, 16(1):20–28, 2000.

[53] J. L. Lowther and C.-K. Shene. Computing with geometry as an undergraduate course: a three-year experience. In *Proceedings of 32nd ACM SIGCSE Technical Symposium*, pages 119–123, 2001.

[54] 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.

[55] F. Mayer, D. Caplan, and K. MacMillan. *SELinux by Example: Using Security Enhanced Linux*. Prentice Hall, 2007.

[56] B. McCarty. *SELINUX: NSA's Open Source Security Enhanced Linux*. O'Reilly & Associates, Inc, 2005.

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

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

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

[60] M. Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[61] T. L. Naps, G. Rößling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, and J. A. Velázquez-Iturbide. Exploring the role of visualization and engagement in computer science education. *SIGCSE Bulletin*, 35(2):131–152, 2002.

[62] B. Neperud, J. Lowther, and C.-K. Shene. Visualizing and animating the winged-edge data structure. *Computers & Graphics*, 31(6):877–886, 2007.

[63] S. Oeltze, D. J. Lehmann, A. Kuhn, G. Janiga, H. Theisel, and B. Preim. Blood flow clustering and applications invirtual stenting of intracranial aneurysms. *IEEE Transactions on Visualization and Computer Graphics*, 20(5):686–701, 2014.

[64] S. O'Hara and B. A. Draper. Introduction to the bag of features paradigm for image classification and retrieval. *arXiv preprint arXiv:1101.3354*, 2011.

[65] O. Pele and M. Werman. Fast and robust earth mover's distances. In *Proceedings of IEEE International Conference on Computer Vision*, pages 460–467, 2009.

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

[67] L. Piegl and W. Tiller. *Curve and Surface Basics*. Springer, 1995.

[68] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch. The state of the art in flow visualisation: Feature extraction and tracking. In *Computer Graphics Forum*, volume 22, pages 775–792, 2003.

[69] A. N. Pressley. *Elementary Differential Geometry*. Springer Science & Business Media, 2010.

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

[71] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *Visualization and Computer Graphics, IEEE Transactions on*, 5(1):47–61, 1999.

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

[73] T. Salzbrunn, H. Jänicke, T. Wischgoll, and G. Scheuermann. The state of the art

in flow visualization: Partition-based techniques. In *Proceedings of Simulation and Visualization Conference*, pages 75–92, 2008.

[74] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[75] 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.

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

[77] T. Schultz. Feature extraction for dw-mri visualization: The state of the art and beyond. *Scientific Visualization: Interactions, Features, Metaphors*, 2:322–345, 2011.

[78] D. W. Scott. On optimal and data-based histograms. *Biometrika*, 66(3):605–610, 1979.

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

[80] J. S. Shimony, A. Z. Snyder, N. Lori, and T. Conturo. Automated fuzzy clustering

of neuronal pathways in diffusion tensor tracking. In *Proceedings of International Society of Magnetic Resonance in Medicine*, volume 10, 2002.

[81] P. Shirley, M. Ashikhmin, and S. Marschner. *Fundamentals of Computer Graphics*. CRC Press, 2009.

[82] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proceedings of 9th IEEE International Conference on Computer Vision*, pages 1470–1477, 2003.

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

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

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

[86] I. Tollis, P. Eades, G. Di Battista, and L. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall New York, 1998.

[87] C. Touma and C. Gotsman. Triangle mesh compression. In *Proceedings of 1998 Graphics Interface*, pages 26–34, 1998.

[88] T. Tuytelaars and K. Mikolajczyk. Local invariant feature detectors: a survey. *Foundations and Trends in Computer Graphics and Vision*, 3(3):177–280, 2008.

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

[90] 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.

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

[92] 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.

[93] M. Yang, K. Kpalma, and J. Ronsin. A survey of shape feature extraction techniques. *Pattern recognition*, pages 43–90, 2008.

[94] H. Yu, C. Wang, C.-K. Shene, and J. H. Chen. Hierarchical streamline bundles.

IEEE Transactions on Visualization and Computer Graphics, 18(8):1353–1367, 2012.

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