A Sketch-Based Interface for Classifying and Visualizing Vector Fields

Jishang Wei*
University of California, Davis

Chaoli Wang†
Michigan Technological University

Hongfeng Yu‡
Sandia National Laboratories

Kwan-liu Ma§
University of California, Davis

ABSTRACT

In flow visualization, field lines are often used to convey both global and local structure and movement of the flow. One challenge is to find and classify the representative field lines. Most existing solutions follow an automatic approach that generates field lines characterizing the flow and arranges these lines into a single picture. In our work, we advocate a user-centric approach to exploring 3D vector fields. Our method allows the user to sketch 2D curves for pattern matching in 2D and field lines clustering in 3D. Specifically, a 3D field line whose view-dependent 2D projection is most similar to the user drawing will be identified and utilized to extract all similar 3D field lines. Furthermore, we employ an automatic clustering method to generate field-line templates for the user to locate subfields of interest. This semi-automatic process leverages the user's knowledge about the flow field through intuitive user interaction, resulting in a promising alternative to existing flow visualization solutions. With our sketch-based interface, the user can effectively dissect the flow field and make more structured visualization for analysis or presentation.

1 INTRODUCTION

Flow visualization is relevant to many areas of science and engineering from understanding the evolution of universe, analyzing weather patterns, to designing more efficient engines for cars. Over the years, many flow visualization techniques and tools have been developed and scientists routinely use them for validating and analyzing the data generated by their simulations or measurements. Despite its widespread uses, one area of flow visualization can use more innovations is visualization of 3D vector fields. For visualizing vector fields, most of the existing techniques are based on particle tracing, and the most commonly used techniques is field-line based visualization, such as streamlines which are curves everywhere tangent to the flow field. One fundamental problem for making streamline visualization is determining the placement and density of streamlines. A related but particularly essential problem for analyzing 3D flow is locating and highlighting flow features of interest in large and complex fields. In this paper, we present a user-centric solution to this problem.

Among different solutions to this problem, automatic clustering plays a vital role towards effective understanding of the flow patterns. There exists a wealth of literature on this topic, which can be divided into two categories: the *voxel-based* and *curve-based* methods. The first category is based on voxel-wise analysis which classifies similar contiguous vectors so that the whole vector field can be divided into several cluster regions. In each cluster, average field lines can be generated to represent the vector characteristics within the region. Alternatively, seed points can be placed to follow important patterns and capture interesting features. In this way, the vector field can be illustrated with sparse field lines. The problem with this voxel-based method is that although a good summary picture of the data set is shown, details may be lost due to averaging quantitative parameters.

The second category is built on curve-based analysis which groups field lines directly. Several methods can be used to represent field lines and measure the similarity between them. For example, each field line can be simply treated as a sequence of traced points. Curves are matched by registering two sets of points following their respective orders along the line. Another way is to match the curves with metrics such as the closest point distance, or the Hausdorff distance. Although this treatment is convenient, the comparison is sensitive to the variation of the underlying flow patterns. A better alternative is to parameterize the curve and represent it with parameters such as length, curvature, and torsion, which can be organized into a new vector that captures the shape feature of the curve. After curve representation and similarity comparison, various clustering approaches, such as k-means or spectral clustering, can be applied to classify the field lines. However, there are still issues with this curve-based method. For instance, most existing solutions are specifically suitable for brain white matter and muscle fiber clustering since these data normally exhibit certain known patterns. Thus, it cannot be easily extended to general vector fields with various flow patterns. Another problem lies in the clustering algorithms themselves. Automatic clustering methods usually aim at dividing curves into different clusters automatically. Nevertheless, for complex vector fields, they could not always achieve satisfactory classification results.

In this paper, we advocate a user-centric approach to field lines classification in which we keep the flow details and explicitly take into account the important role that the user plays in the visual data exploration process. Our approach is achieved through an intuitive sketch-based interface and straightforward interaction: the user simply sketches a 2D curve and this input curve will be used to brush similar streamlines in 3D. A typical scenario is when the user is examining streamlines of a complex vector field and is interested in certain line pattern, she can then sketch a curve in the 2D interface close to what is observed under the current view. The input curve will be used to match and retrieve all similar 3D streamlines automatically. Note that our interface allows fuzzy specification of the input curve and our algorithm is able to identify the most similar streamlines. By introducing user interaction, streamlines can be clustered in a way that is meaningful to the users. In addition, we also present a solution that applies similar user input to query streamline templates produced from an automatic clustering algorithm. We believe that our semi-automatic approach provides an alternative for scientists to understand vector fields where simple yet informed user input plays a central role.

2 RELATED WORK

One of the most important goals in flow visualization is to display the vector field in a clear and meaningful way, in which flow characteristics are carefully presented and thus can be easily perceived.

*e-mail: jswei@ucdavis.edu
†e-mail: chaoliw@mtu.edu
‡e-mail: hyu@sandia.gov
§e-mail: ma@cs.ucdavis.edu

Different approaches have been taken to study this topic.

A key issue that affects the quality of streamlines representation is the seeding strategy and streamlines placement [11] because straightforward solutions would easily lead to clutter. Verma et al. [20] presented a seed placement strategy to capture flow patterns around the critical points and to provide sufficient coverage in non-critical regions. Ye et al. [23] presented a method for streamline placement in 3D flow field. Their aim is to present all significant flow patterns with maximum coverage and minimum clutter. Chen et al. [3] introduced a technique for the placement of streamlines in 2D and 3D steady vector field. In their approach, a new metric for local similarity measurement among streamlines was introduced to guide field line generation. This metric involves statistical information of streamline shape and direction as well as the Euclidean distance of each pair of curves. Li et al. [8] presented an image-based seeding strategy which is used for 3D flow visualization. Their algorithm tries to control scene cluttering by placing seeds in the image plane in such a way that the depths and structures of streamlines can be clearly presented. Li et al. [7] also introduced another method for streamline placement which is different from [8] in that its goal is to generate representative and illustrative streamlines in 2D vector fields to enforce visual clarity and evidence. They used a minimum amount of streamlines to describe the underlying patterns by only creating new lines at places with flow characteristics that have not shown yet. In [17], Spencer et al. described an automatic streamline seeding algorithm for vector fields defined on surfaces in 3D space. This algorithm generates evenly-spaced streamlines quickly and efficiently for any general surface-based vector field.

Clustering is another key technique to generate meaningful visualization for representing complex flow patterns. In voxel-based clustering, seed points are placed in individual clusters to capture important flow patterns. A holistic overview in a coarser definition is exhibited along with clutter-free rendering. Heckel et al. [6] introduced a top-down clustering method by splitting groups of voxels iteratively. This approach allows more than one level in the hierarchy to be visualized simultaneously. Telea et al. [18] proposed a bottom-up method by merging neighboring groups of voxels with the highest similarity in an iterative manner. Simplified vector field visualization results can be obtained at different levels of details. Du et al. [4] presented a clustering technique based on the centroidal Voronoi tessellation (CVT) to separate the vector field into a fixed number of groups. McKenzie et al. [10] extended the work in [4] by employing different error metrics in the variational clustering.

Another category of vector field clustering is based on direct clustering of field lines. This approach is most commonly found in visualizing diffusion tensor imaging (DTI) data such as brain white matter and muscle fibers [12]. In general, these algorithms all share the steps of first defining a similarity metric between the trajectories, and then employing an algorithm for clustering according to the established similarity measurement. For instance, Shimony et al. [16] tested several distance metrics that measure the distance between tracks. They used the fuzzy c-means algorithm for clustering. Brun et al. [1] compared pairwise fiber traces in dimension-reduced Euclidean feature space to create a weighted, undirected graph which is partitioned into coherent sets using the normalized cut. O'Donnell et al. [13] utilized the symmetrized Hausdorff distance as the similarity measurement among trajectories and achieved spectral clustering using the Nyström method and the k-means algorithm in the embedding space. Tsai et al. [19] constructed tract distances between fiber tracts from dual-rooted graphs where both local and global dissimilarities are taken into account. The considered distance is then incorporated in a locally linear embedding framework and clustering is performed using the k-means algorithm. Curve modeling has also been utilized in clustering. For example, Maddah et al. [9] defined a spatial similarity measure between curves for a supervised clustering algorithm. Expectation-maximization (EM) algorithm is used to cluster the trajectories in the context of a gamma mixture model.

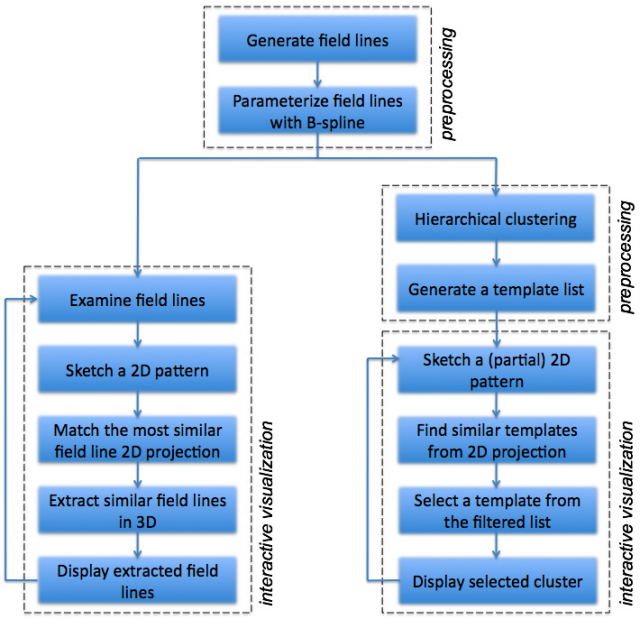


Figure 1: The sketch-based vector field visualization process. The user can use sketching to find field lines (left branch) or field-line clusters (right branch) of interest.

3 Overview

Figure 1 illustrates the flowchart of our vector field sketching and clustering process. Our approach introduces user interaction into the clustering. We assume that we are given enough field lines that could capture as many features as possible in the vector field. For instance, we can randomly place a large number of seed points to generate field lines. We point out that our method can work with existing seeding strategies that capture different aspects of flow features. The resulting field lines may provide valuable information for purposeful sketching. Our approach consists of two different directions: one uses sketching to find field lines and the other uses sketching for querying field line template.

3.1 Sketching for Field Line Clustering

For field line clustering, our approach includes two steps: 2D curve selection [22, 2] and similar 3D field lines clustering [14]. The user's input is constrained in the 2D space. Thus all field lines are projected into the 2D space for matching with the input curve. The projection corresponds to a user-specified viewpoint where features of interest may exist. Based on these projected curves, we utilize the *string matching* approach to find the field line that is most similar to the input. Specifically, we approximate the user's drawing and the projected field lines by an arc length parameterized cubic B-spline. The shape features are extracted by sampling the curvature at equal arc length intervals along the curve. Then, we measure the similarity between two curves using the *edit distance*.

After getting the 2D curve resembling the input pattern, we use its original 3D field line as the reference for 3D field line clustering. The 3D similarity measurement is similar to that in the 2D curve selection. The only difference is that one 3D curve should be represented by both curvature and torsion. Identified similar 3D field lines will be highlighted with different colors and/or opacities. They can also be removed from the rendering so that the user can

continue to explore the remaining field by changing the view and sketching interested patterns observed. New groups of similar field lines will be identified. After a few iterations, the entire vector filed would be partitioned into different clusters. The user can examine one group at a time, or combine several groups to find their relations. Alternatively, representative field lines from each group can be used for selective display so that a less cluttered visualization can be realized while distinct flow patterns are exhibited.

3.2 Sketching for Template Query

For field line template query, a list of clusters is pre-generated using a hierarchical clustering algorithm during preprocessing. The algorithm allows the user to change the number of clusters interactively at runtime. For each cluster, a view-dependent field line that displays the maximum amount of projection information will be used as the template to represent the corresponding cluster. The user sketches a partial 2D pattern and our algorithm dynamically reorders the pre-genereated field line templates according to their similarities. This step is similar to the 2D curve selection described in Section 3.1. This similarity comparison helps narrow down a potentially large number of templates to only a few most similar ones for user selection. As the user keeps drawing the pattern, the template reordering becomes more accurate. The user can simply select one from the filtered template list anytime during her drawing and the corresponding filed line cluster will be highlighted in the display. Unlike field line clustering, the reason that we can perform dynamic reordering for template query is due to the limited number of templates generated, which normally is set by the user and is much smaller than the total number of input field lines.

4 Sketch-Based Clustering

4.1 Representation of 2D User Sketching

Although drawing 3D curves is possible, we constraint the user input in a 2D interface because it is more intuitive and convenient for the user to sketch 2D curves. A meaningful representation of the input drawing is required for the following 2D curve selection and 3D curve clustering. We use a descriptor that includes geometric parameters such as the curvature along the curve. The curvature is defined as an instantaneous rate of change of the angle with respect to arc length. The main advantage of this representation is that it is invariant with respect to rotation and translation.

In practice, direct handling the user's sketching leads to some problems related to image rasterization. Moreover, small changes cannot be preserved in this case. To address this, we smooth the user's sketching by sampling it at certain intervals and remodel it with the cubic B-spline. As such, the curve is defined as a function of parameter t in the range of $[t_{\min}, t_{\max}]$, where $t_{\min}$ and $t_{\max}$ correspond to the start and end of the curve. That is,

$$\overrightarrow{V}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}.$$

To compute the line's curvature, the curve needs to be reparameterized by the arc length s in the range of $[0, L]$ where L is the total length along the curve, i.e.,

$$\overrightarrow{V}(s) = \begin{pmatrix} x(s) \\ y(s) \end{pmatrix}.$$

Let $\overrightarrow{V}(s)$ and $\overrightarrow{V}(t)$ denote the same curve, in which $\overrightarrow{V}(s) = \overrightarrow{V}(t)$ implies a relationship between t and s. We apply the chain rule to obtain

$$\frac{d\overrightarrow{V}(t)}{dt} = \frac{d\overrightarrow{V}(s)}{ds}\frac{ds}{dt}.$$

Then we have

$$\left|\frac{d\overrightarrow{V}(t)}{dt}\right| = \left|\frac{d\overrightarrow{V}(s)}{ds}\right|\left|\frac{ds}{dt}\right| = \frac{ds}{dt},$$

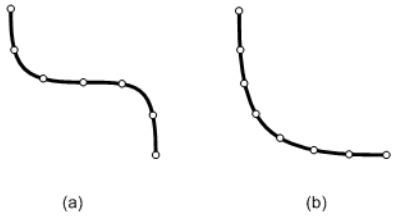


Figure 2: Examples of the sign on curvature. In (a), the first four points (from left to right) have the positive sign while the rest three points have the negative sign. In (b), all points have the positive sign.

where $\left|\frac{d\overrightarrow{V}(s)}{ds}\right| = \frac{\left|d\overrightarrow{V}(s)\right|}{|ds|} = 1$ and $\frac{ds}{dt}$ is nonnegative as the arc length s increases with the increase of t. From this equation, we can get the relationship between s and t by integration

$$s = \int_{t_{min}}^{t} \left|\frac{d\overrightarrow{V}(\tau)}{dt}\right| d\tau,$$

when $t = t_{min}$, $s = 0$; and when $t = t_{max}$, $s = L$. Given t, we can determine the corresponding arc length s from the integration. However, what we need is the inverse problem. Given an arc length s, we want to know the corresponding t. To solve this, we employ numerical methods to compute $t \in [t_{min}, t_{max}]$ given the value of $s \in [0, L]$. We treat

$$\frac{dt}{ds} = \frac{1}{\left|\frac{d\overrightarrow{V}(t)}{dt}\right|}.$$

The above equation may be solved numerically with any standard differential equation solver such as the Runge-Kutta method.

Let $\overrightarrow{T(s)}$ denote the unit vector tangent to this curve and θ be the angle change of two consecutive vectors. Expressed in terms of the derivatives of $x(s)$ and $y(s)$ with respect to the arc length s, the curvature (κ) at point s is then given by

$$\kappa(s) = \frac{x'(s)y''(s) - y'(s)x''(s)}{\left(x'(s)^2 + y'(s)^2\right)^{\frac{3}{2}}},$$

where the absolute value of κ is larger when the curve is turning rapidly and smaller when the curve is relatively straight. In our system, instead of using the exact definition of curvature, we use the angular difference between the tangent vectors at two consecutive points of the curve [5]. The benefit of using the angular difference is that it is easy to understand with clear physical meaning. Besides, it is simple to set in the cost function for string matching.

Another key issue is how to set the sign of angular difference. In our application, at the starting point the sign of angular difference is assigned as positive. In the following steps, if angular difference of two consecutive vectors changes from clockwise to counterclockwise or vise versa, the sign is changed to the opposite. For instance, in Figure 2(a), the angular difference at the first four points are positive and those at the last three points are negative since the proceeding direction of the curve changes from counterclockwise to clockwise. In Figure 2(b), the angular difference on all points are positive since the curve moves along the counterclockwise direction all the time.

At this point, we have already constructed a signature for one curve composed of an ordered set of accumulated curvatures along the line at equal intervals:

$$f = (c_1, c_2, \ldots, c_r).$$

However there is still one problem remaining. During point sampling along the curve, some points with large curvature may be

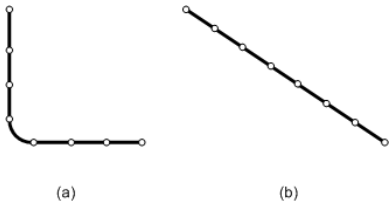


Figure 3: Sampling points on two different curves yields the same feature vector since in (a), the corner is not sampled. Taking into account the accumulated curvature can capture the difference between the two curves.

missed, which is important in describing the curve. For example, in Figure 3, we show a sequence of points sampled along the line in (a) and (b), respectively. Because the turning point in (a) is not sampled, the feature signature for both (a) and (b) will be exactly the same as $f = (0,0,0,0,0,0)$. To overcome this problem, we apply the *accumulated curvature* in which angular differences along the curve are summarized first and then the total angular differences for that curve segment is used.

According to this feature generation method, different field lines would have different numbers of curvatures in the feature representation. Actually, all the information about the curve such as the total arc length, angle change, and vertex order, are saved in the signature. With this representation, we will select the most similar curves to the user's input and then cluster all similar field lines in the 3D space.

4.2 Matching 2D Field Line Projections

In our system, the user can observe the vector field from different points of view. At a certain viewing direction, the user may find some pattern she is interested in and she can sketch this curve pattern in the given 2D interface. The system will find similar field line projections in the 2D space under the current view.

4.2.1 String Matching

We use the string matching approach to find similar curves based on the user's input. A *string* is composed of primitives which are *N*-dimensional vectors of numerical values. To measure the difference between strings, we employ the *edit distance*. The *edit distance*, introduced by Wagner and Fisher [21], is a metric for measuring the amount of difference between two sequences. The edit distance between two strings is given by the minimum number of operations needed to transform one string into the other, where an operation is an insertion, deletion, or substitution of a single primitive. For the 2D curve representation, every primitive vector composing one string is only one-dimensional which is the curvature. Our basic idea is to represent both the input line and the field line's 2D projection as strings using the method introduced in Section 4.1. In this way, the curve matching problem can be transformed into string matching, which is implemented using dynamic programming. The reference curve will be the user's sketching and all the other lines will be matched accordingly to identify the most similar one.

One critical issue with string matching is the definition of the cost function. There are two contradictory considerations in our application. One of them is to enhance the total arc length of each curve. For instance, we will not classify curves with significant different lengths into one group because curves with different lengths in the vector field usually indicate different patterns. In other words, similar lines have similar lengths. The other consideration is to insert or delete one primitive whenever it is necessary. That means when the curvatures at some pair of points make a significant difference, we should delete or insert a point instead of only substituting corresponding curvatures and count the difference in the final cost. With this consideration, we define a cost function as follows

$$f_{sub} = \kappa_{xi} - \kappa_{yj}, \quad f_{del} = f_{ins} = c \max f_{sub};$$

where f_{sub}, f_{del}, and f_{ins} are the costs for the substitution, deletion, and insertion, respectively; κ_{xi} and κ_{yj} are curvatures of the ith and jth sample points on curves x and y, respectively; $c \in (0,1)$ is a constant and is set to $2/3$ in this paper. The heuristic is that the deletion or insertion operation must be greater than half of the cost of substitution operation. Otherwise, each pair of strings will be matched using the deletion or insertion operation only.

4.2.2 Repetition, Scale, and Tolerance in Patterns

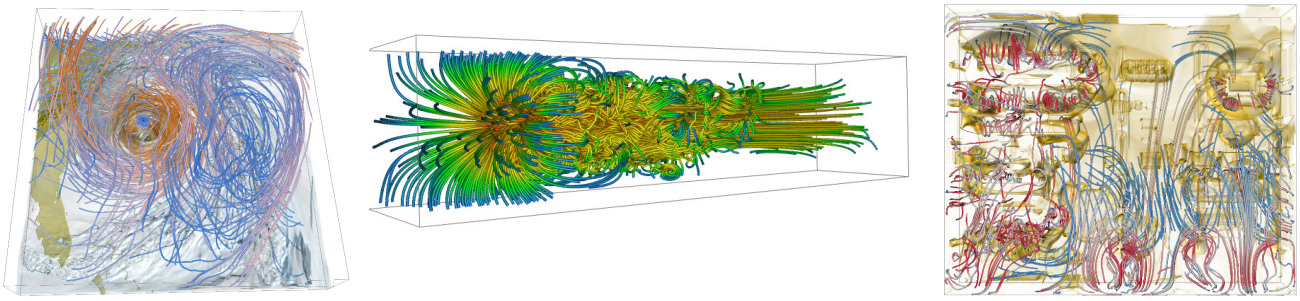
If streamlines are traced very long due to the underlying flow pattern, the total length and angle changes of each field line can be very large. As a consequence, it will be difficult for the user to characterize the whole curve. At the same time, the number of string primitives for describing each line will increase, which would demand more time for curve matching. As a matter of fact, we observe that many feature patterns are repetitive along the curve. Therefore, only the characteristic part of the curve needs to be matched instead of the whole line. A simple example is a circular field line with a large number of loops. In this case, the user would prefer to sketch one loop to catch this feature field line rather than sketching many loops. In our work, we propose a simple yet effective way to extract line patterns for matching. According to the angle changes along the curve, only sample points at which the absolute value of total angle change is less than a certain value are considered. In our application, we set this value to 720°. In this manner, both the line patterns on curves or circles can be successfully captured.

The scale of the field lines indicates its relative size. The change of scales for certain flow patterns is an important aspect and should be taken into consideration in curve matching as well. In some cases, flow patterns of small size are just noise, but in others, they are probably important areas in the vector field. For example, eddies/vortices/swirling zones at a particular scale in practice, could be either good or bad indications. To incorporate this, we normalize all points on the curve into the range of $[0,1]$ and store its magnitude during the B-spline parameterization. The magnitude which indicates its relative scale will serve as an additional parameter in curve matching. In this way, the user is able to find the same shape of features but in different scales.

Another important parameter is the similarity tolerance. Instead of setting the extent beforehand and determining which range of similar field lines should be assigned to one group, we allow the user to change the similarity tolerance on the fly. By varying this parameter, the user can either display only the most similar streamlines or most of the streamlines, and everything else in between. This also makes it possible to show how different groups of field lines are similar to each other as the tolerance parameter varies. In addition, we can also use the total angle change and curve length as the tolerance. The assumption here is that similar flow patterns have similar total angle changes and curve lengths.

4.3 Matching 3D Field Lines

Only matching curves in 2D does not complete our work because similar field lines in 3D are likely to have quite different shapes after projected onto the 2D viewing plane. Our goal is to find all similar field lines, and therefore, we need to classify field lines in the 3D space. We point out that the 2D curve matching method can be extended to 3D. The difference lies in the representation of 2D and 3D curves. In 3D, a curve can be uniquely represented by its curvature and torsion while in 2D, using only the curvature information suffices. The curvature (κ) and torsion (τ) in 3D can be represented as



(a) (b) (c)

Figure 4: (a)-(c) show streamlines traced in the hurricane, plume, and computer room data sets, respectively. The velocity magnitudes are mapped to the streamline colors. For the hurricane and computer room data sets, the scalar fields are also volume rendered to provide the context information.

$$\kappa(s) = \frac{K^{\frac{1}{2}}}{\left(x'(s)^2 + y'(s)^2 + z'(s)^2\right)^{\frac{3}{2}}}$$

and

$$\tau(s) = \frac{\left|\det\left(\begin{bmatrix} x' & y' & z' \\ x'' & y'' & z'' \\ x''' & y''' & z''' \end{bmatrix}\right)\right|}{K},$$

where

$$\begin{aligned} K &= (y'(s)z''(s) - y''(s)z'(s))^2 + \\ &= (z'(s)x''(s) - z''(s)x'(s))^2 + \\ &= (x'(s)y''(s) - x''(s)y'(s))^2. \end{aligned}$$

The curvature and torsion at each sample point can be organized into a two-dimensional vector, which serves as a primitive in string matching. Similar to the 2D case, the cost functions now become

$$f_{sub} = (\kappa_{xi} - \kappa_{yj}) + (\tau_{xi} - \tau_{yj}), \quad f_{del} = f_{ins} = c \max f_{sub};$$

where κ_{xi} and τ_{xi} are curvature and torsion values of the ith sample point on curve x respectively, and so are κ_{yj} and τ_{yj} at the jth sample point on curve y. Again, we set $c = 2/3$.

5 Sketch-Based Template Query

For automatic clustering, we use a typical agglomerative hierarchical clustering method. Before clustering, each field line is considered as a separate cluster. Then, we perform a bottom-up clustering algorithm iteratively such that in each iteration, the two most similar lines are merged into a larger cluster. This merging process repeats until we arrive at a single cluster that represents the whole set of field lines. A binary tree is generated during this clustering process indicating which two clusters are merged at each iteration. Each tree node has a level attribute indicating at which stage the cluster is created. By default, all leaf nodes have the level of 0 and the root has the level of $N-1$, where N is the number of field lines traced from the flow data set. All the initial field lines are stored in the leaf nodes.

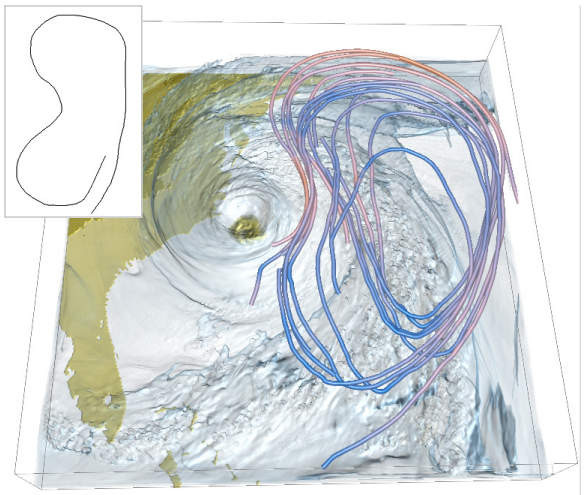
The key step of this clustering algorithm is to evaluate the similarity of two clusters and merge them. In our implementation, each cluster is associated with a representative string. Initially, for each field line, the representative string is its corresponding string, which is constructed in the same way as described in Section 4.2.1. The similarity evaluation of two clusters compares their representative strings. When merging two clusters into a new one, the representative string for the new cluster is the mean string of the merged clusters. The mean string is calculated using the approach presented in [15]. The clustering is performed at the preprocessing stage.

During runtime visualization, we first select the number of clusters by choosing the level in the binary tree. Then, the selected clusters are displayed using the projections of their representative field lines. The selection of the representative field line for a cluster is view-dependent. Give a cluster, we traverse its corresponding binary tree node to obtain all its leaf nodes where the initial field lines are stored. Among them, we select a line with the maximum viewpoint quality as the representative line of a cluster. We measure the viewpoint quality of a line using a heuristic approach. We first sample the 2D projection of the field line, and then accumulate the angles of the two consecutive sample line segments, where a larger angle represents more information shown in the 2D projection and therefore a higher viewpoint quality. In practice, we precalculate the viewpoint qualities of a field line by sampling the viewing sphere with a constant angle spacing (say 5°). At runtime, the quality of a field line from a viewing direction is looked up from the closest sampled viewpoint. The measurement of the similarity between the pattern sketched by the user and the projected representative field lines of the clusters follows the same method described in Section 4.2. The measuring is performed in real time as the user sketches a pattern. The most similar clusters are dynamically updated and listed for user selection.

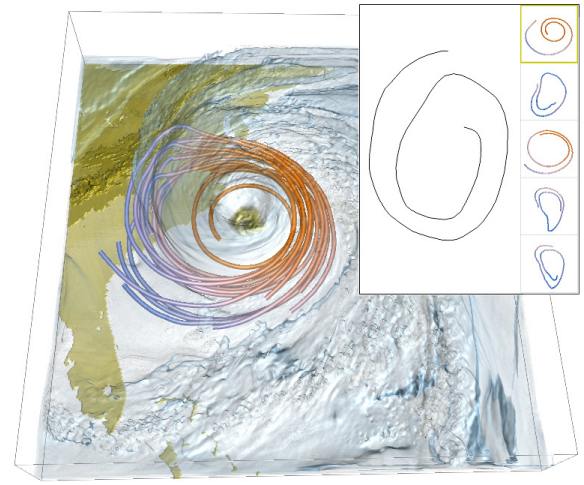
6 Results and Discussion

6.1 Results

We experimented with our user-centric approach on several flow data sets. The first one is the velocity vector field of the hurricane Isabel data set, provided by the National Science Foundation and the National Center for Atmospheric Research (NCAR). The second one is the solar plume velocity vector field, provided by the scientists at the NCAR. The third one is the flow field in a computer room. Figure 4 shows the streamlines generated in the three data sets respectively by random seeding in the vector field and tracing with the Runge-Kutta method. Clearly, all rendered images contain a great deal of clutter. In the following, we demonstrate how our sketch-based interface helps the users specify line patterns and cluster the flow fields. We also present results that use sketch-based interface for template query where the corresponding clusters are pre-generated using the automatic clustering algorithm.



(a) sketch-based clustering



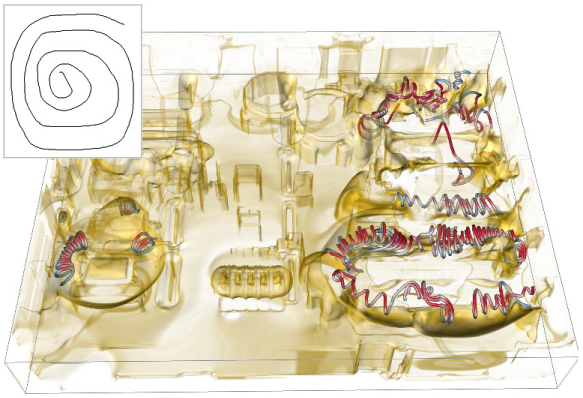
(b) sketch-based template query

Figure 5: (a) The user sketches an ellipse pattern and the matching streamlines are displayed. (b) The user sketches a circular pattern and the most similar five templates are displayed from top to bottom. The user then picks the one on the top and the selected cluster is displayed. In the figure, the velocity magnitudes are mapped to the streamline colors.

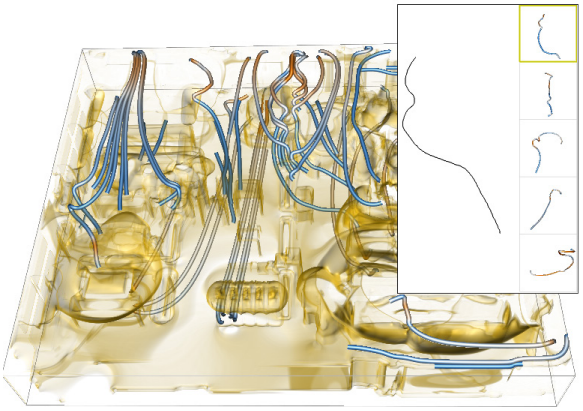
6.1.1 Sketch-Based Clustering and Template Query

Examples of the user sketching and the matching streamlines are displayed in Figure 5 (a) and Figure 6 (a) for the hurricane and computer room data sets, respectively. The experiments show that our 2D pattern matching and 3D streamline clustering algorithm work pretty well for different kinds of patterns: straight or curved, long or short. The user can draw close to what have been observed and similar streamlines will be displayed interactively. This ability is very useful for exploring a complex flow data and isolating one interesting cluster at a time. The user is able to understand the flow features interactively through customized pattern specification.

In Figure 5 (b) and Figure 6 (b), we show results with the user sketching and the matching templates for the two data sets. Our system dynamically identifies the most similar templates as the user sketches the pattern on the fly. The most similar ones are displayed for user choices. Whenever the user finds the desired template, she can simply click the template and the corresponding cluster is displayed. In this mode, the sketching is used as the querying interface.



(a) sketch-based clustering



(b) sketch-based template query

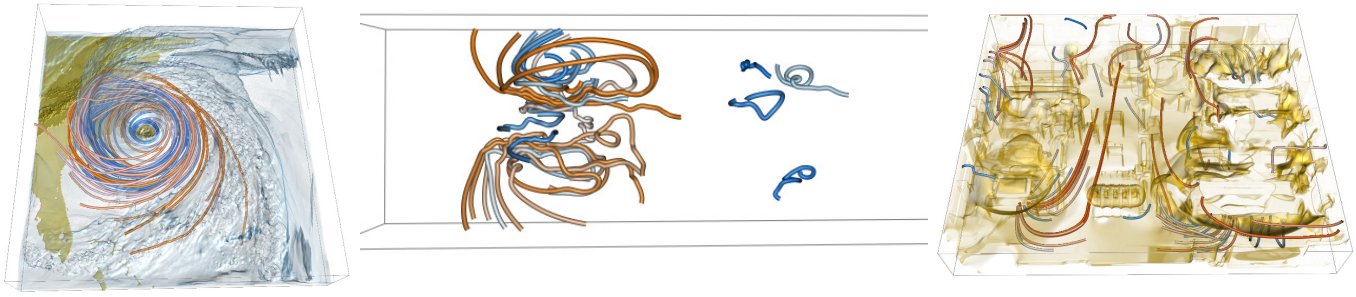
Figure 6: (a) The user sketches a spiral pattern and the matching streamlines are displayed. (b) The user sketches a long-tail pattern and the most similar five templates are displayed from top to bottom. The user then picks the one on the top and the selected cluster is displayed. In the figure, the velocity magnitudes are mapped to the streamline colors.

6.1.2 Scale and Similarity Threshold Control

Streamlines sharing the similar shape may have different scales. To allow the user to explore the scale, we provide a slider where the user can change the magnitude threshold to brush similar streamlines at different scales. Figure 7 shows examples with the three data sets. As we can see, similar streamlines now can be differentiated through their varying scales. This capability is useful as the user may pay attention to larger scale features first and then focuses on smaller scale features in local regions. Figure 8 shows examples with the three data sets with varying similarity tolerance values as computed in section 4.2.2. As the user varies the tolerance value, she can have an intuitive understanding of how streamlines with different degrees of similarity differ from each other and how they are distributed over the space.

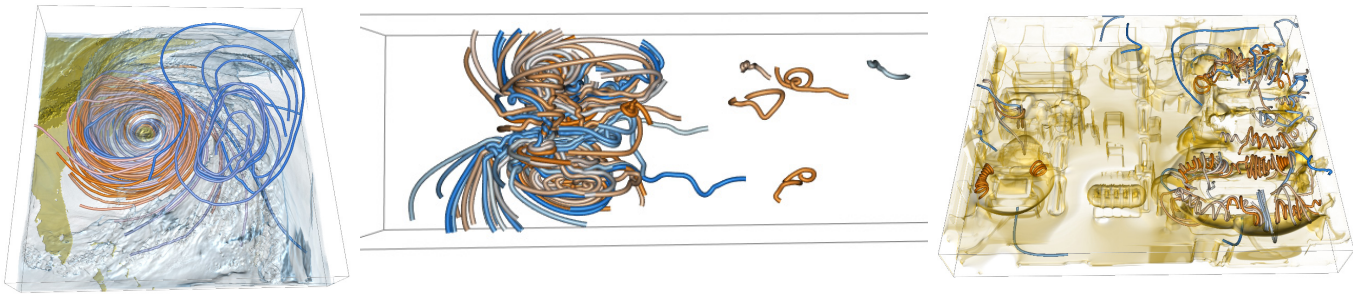
6.1.3 Visualization of Multiple Clusters

In our system, the user can iteratively explore the flow data and brush the streamlines to identify multiple clusters in order. Once the user is satisfied with her exploration results, the system can display multiple clusters with different color, opacity, or rendering styles to differentiate them. We can also selectively display a subset of streamlines from each cluster for a less cluttered view.



(a) (b) (c)

Figure 7: Examples showing the sketching of similar streamlines with different scales. large-scale streamlines are orange and small-scale ones are blue. The corresponding patterns sketched for (a)-(c) are the circular, curly, and long-tail patterns, respectively.



(a) (b) (c)

Figure 8: Examples showing the sketching of similar streamlines with different tolerance. More similar streamlines are orange and less similar ones are blue. The corresponding patterns sketched for (a)-(c) are the circular, curly, and spiral patterns, respectively.

Figure 9 shows these visualizations with the three data sets, respectively. Compared with the original rendering with all streamlines displayed, Figure 9 demonstrates that we can now generate a clearer and more meaningful picture to reveal the different flow patterns.

6.2 Discussion

During the preprocessing, we trace a large number of field lines from the input vector field. We then parameterize these field lines using the B-spline. At run time, for several hundreds to a few thousand of field lines with an average of 50 steps for each line using the B-spline parameterization, our system works interactively with user exploration and is able to find similar field lines in real time. The algorithm accepts fuzzy input in the sense that the user can sketch patterns close to field line 2D projections. However, in some cases, the system may fail to recognize the desired pattern because what the user sketches may belong to only a small part of the entire pattern (the large part not drawn is not clearly visible). We need to evaluate the sensitivity of input pattern to the correctness of field lines found. Another aspect that needs improvement is the cost function for string matching. We currently use a fixed value for constant c. For a better similarity matching, the cost function should be adaptive to the characteristics of the flow fields.

7 Conclusions

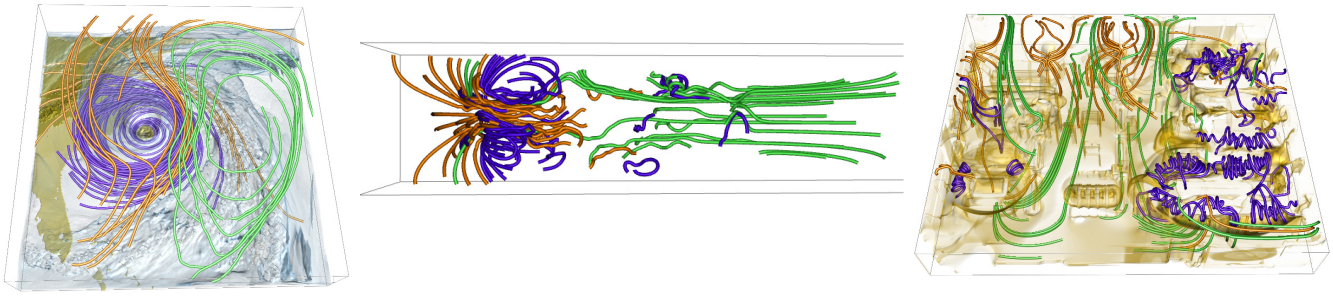
Three-dimensional vector field visualization stays a challenging problem. The sketch-based technique we have introduced complements existing methods by offering a new way to probe and dissect a flow field. It dictates a user centered and domain knowledge directed process, which we believe is key to the understanding of large and complex flow fields. While fully automated methods are absolutely required for handling large data sets, our technique suggests scientists to rethink how they may approach flow visualization problems to validate their modeling or measurement and to possibly reveal some previously hidden flow structures and patterns in their data. For further work, we will involve scientists in the evaluation of our technique. To support real-world applications, the technique must be extended to handle time-varying vector fields, that is to extract and track complex flow features over time. We will also develop advanced interfaces and interaction techniques for such a user centric approach to flow visualization. In particular, sketching and brushing may be done creatively and quite naturally through touch and stereoscopic interfaces.

Acknowledgements

This research was supported in part by the U.S. National Science Foundation through grants OCI-0325934, OCI-0749217, CNS-0551727, CCF-0811422, OCI-0749227, OCI-0950008, CCF-0938114 and OCI-0850566, and the U.S. Department of Energy through the SciDAC program with Agreements No. DE-FC02-06ER25777 and DE-FG02-08ER54956.

References

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



(a) (b) (c)

Figure 9: (a)-(c) show the simultaneous display of multiple clusters derived from the sketch-based clustering. Each cluster is illustrated with a different color. A selective subset of streamlines is displayed for each cluster. Compared with the corresponding images in Figure 4, this visualization shows a much clearer picture of the flow patterns.

[2] H. Bunke and U. Buhler. Applications of approximate string matching to 2d shape recognition. *Pattern Recognition Letters*, 26(12):1797–1812, 1993.

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

[4] Q. Du and X. Wang. Centroidal Voronoi tessellation based algorithms for vector fields visualization and segmentation. In *Proceedings of IEEE Visualization Conference*, pages 43–50, 2004.

[5] A. Gray, E. Abbena, and S. Salamon. *Modern differential geometry of curves and surfaces with Mathematica*. CRC Press, 2006.

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

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

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

[9] M. Maddah, W. E. L. Grimson, S. K. Warfield, and W. M. Wells. A unified framework for clustering and quantitative analysis of white matter fiber tracts. *Medical Image Analysis*, 12(2):191–202, 2008.

[10] A. McKenzie, S. Lombeyda, and M. Desbrun. Vector field analysis and visualization through variational clustering. In *Proceedings of Eurographics-IEEE VGTC Symposium on Visualization*, pages 29–35, 2005.

[11] T. McLoughlin, R. S. Laramee, R. Peikert, F. H. Post, and M. Chen. Over two decades of integration-based, geometric flow visualization. In *Eurographics State of the Art Report*, pages 73–92, 2009.

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

[13] L. O'Donnell and C.-F. Westin. White matter tract clustering and correspondence in populations. In *Proceedings of International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 140–147, 2005.

[14] W. Rodriguez, M. Last, A. Kandel, and H. Bunke. 3-dimensional curve similarity using string matching. *Robotics and Autonomous Systems*, 49(3-4):165–172, 2004.

[15] G. Sánchez, J. Lladós, and K. Tombre. A mean string algorithm to compute the average among a set of 2d shapes. *Pattern Recognition Letters*, (23):203–213, 2002.

[16] J. S. Shimony, A. Z. Snyder, N. Lori, and T. E. Conturo. Automated fuzzy clustering of neuronal pathways in diffusion tensor tracking. In *Proceedings of International Society for Magnetic Resonance in Medicine*, pages 453–456, 2003.

[17] B. Spencer and M. W. Jones. Evenly-spaced streamlines for surfaces: An image-based approach. *Computer Graphics Forum*, 28(6):1618–1631, 2009.

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

[19] A. Tsai, C.-F. Westin, A. O. Hero, and A. S. Willsky. Fiber tract clustering on manifolds with dual rooted-graphs. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–6, 2007.

[20] V. Verma, D. Kao, and A. Pang. A flow-guided streamline seeding strategy. In *Proceedings of IEEE Visualization Conference*, pages 163–170, 2000.

[21] R. Wagner and M. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

[22] H. J. Wolfson. On curve matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(5):483–489, 1990.

[23] X. Ye, D. Kao, and A. Pang. Strategy for seeding 3d streamlines. In *Proceedings of IEEE Visualization Conference*, pages 471–478, 2005.