FlowTour: An Automatic Guide for Exploring Internal Flow Features

Jun Ma* James Walker† Chaoli Wang‡ Scott A. Kuhl§ Ching-Kuang Shene¶

Michigan Technological University

ABSTRACT

We present FlowTour, a novel framework that provides an automatic guide for exploring internal flow features. Our algorithm first identifies critical regions and extracts their skeletons for feature characterization and streamline placement. We then create candidate viewpoints based on the construction of a simplified mesh enclosing each critical region and select best viewpoints based on a viewpoint quality measure. Finally, we design a tour that traverses all selected viewpoints in a smooth and efficient manner for visual navigation and exploration of the flow field. Unlike most existing works which only consider external viewpoints, a unique contribution of our work is that we also incorporate internal viewpoints to enable a clear observation of what lies inside of the flow field. Our algorithm is thus particularly useful for exploring hidden or occluded flow features in a large and complex flow field. We demonstrate our algorithm with several flow data sets and perform a user study to confirm the effectiveness of our approach.

1 INTRODUCTION

For large and complex 3D flow fields, carefully placed or selected streamlines can effectively reveal characteristic flow information. As such, streamline placement or selection has been well studied. However, due to the projection of 3D streamlines to 2D images, streamline occlusion and clutter cannot be eliminated, which hinders the visual comprehension of essential flow features, especially hidden or occluded ones. Algorithms have been developed for finding characteristic viewpoints to help users better find underlying flow patterns. However, most existing solutions of viewpoint selection for volume and flow data are restricted to *external* viewpoints, excluding potentially more effective observation with *internal* viewpoints.

In this paper, we present FlowTour, a novel framework that provides an automatic guide for exploring internal flow features. Unlike external viewpoints which are normally selected from the volume's bounding sphere and look at the center of the volume, internal viewpoints are more difficult to determine since they can be placed anywhere inside the volume and look at any points of interest. Our algorithm encompasses feature identification, streamline placement, viewpoint selection and tour generation into a single, unified framework. Since our viewpoint selection places its focus on internal viewpoints, the final tour going through multiple critical regions is similar to a roller coaster tour in an amusement park which flies through the scene. In this way, we give users closeup views of the flow field for detailed observation of hidden or occluded internal flow features and patterns.

Previous work on view selection and path generation mostly focused on scalar data, leaving vector data unexplored. To the best of our knowledge, our work is the first that aims to design a tour for examining internal flow features. The main contributions of our work are the following: First, we present a novel method for finding characteristic viewpoints for internal exploration of a flow field based on the analysis and extraction of critical regions. Second, we generate a tour which traverses all selected viewpoints in a smooth and efficient way. This provides users with an immersive viewing experience which helps them gain a more comprehensive and detailed picture about the flow data. Third, we perform a user study to show the effectiveness of our FlowTour solution and confirm the benefits of including internal viewpoints in the design.

*e-mail: junm@mtu.edu

†e-mail: jwwalker@mtu.edu

‡e-mail: chaoliw@mtu.edu

§e-mail: kuhl@mtu.edu

¶e-mail: shene@mtu.edu

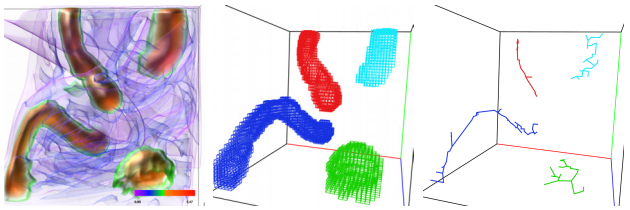
2 RELATED WORK

Seed placement and streamline selection are two major directions for streamline visualization. Examples for seed placement include image-guided placement [17], evenly-spaced seeding [6], feature-guided seeding [19], farthest seed placement [12], image-based placement [8], dual seeding [13], surface seeding [14] and entropy-based seeding [21]. For streamline selection, several research efforts utilized concepts from information theory to quantify the importance of streamlines for selection [11, 7, 10]. All these methods [11, 7, 10] are view-dependent and the work recently presented by Tao et al. [16] utilizes a unified information channel between streamlines and viewpoints to select streamlines in a view-independent manner.

Viewpoint selection is an important problem in scientific visualization. Viewpoint entropy was introduced by Vázquez et al. [18] for viewpoint selection for polygonal models. Similar ideas based on information theory were later applied to viewpoint selection for volume visualization [1, 15, 5, 20], mesh saliency [2] and flow visualization [16]. After finding the best viewpoints, camera path planning was also studied to generate a good path traversing all selected viewpoints [5, 20, 2, 16].

3 ALGORITHM OVERVIEW

Our approach includes three major stages: critical region identification and skeleton-based seeding, viewpoint creation and quality evaluation, and viewpoint selection and tour generation. Each stage consists of several substeps. At the first stage, given an input 3D flow field, we compute its *entropy field* [21] and identify *critical regions* which correspond well to interesting flow features and patterns such as the vicinities of critical points. We detect large critical regions for the flow field and compute an isosurface and a skeleton for each of them. A *skeleton-based seeding* algorithm is carried out to purposefully generate a set of streamlines for the subsequent viewpoint evaluation and tour design. At the second stage, we convert the isosurfaces of critical regions into triangle meshes and initialize vertices on each simplified mesh as candidate viewpoints associated with the critical region. A series of *offset viewpoints* with different zooming levels is computed for each viewpoint to construct a *viewpoint set*. We evaluate the quality of viewpoint by considering how much information of the streamlines seeded from the corresponding critical region could be revealed. We also take into account *foreground streamline occlusion* and *background streamline noise* in the evaluation. At the third stage, we select one viewpoint with the highest quality as the representative for each viewpoint set. We then pick several best viewpoints from all the representative viewpoints for the corresponding critical region. The final



(a) (b) (c)

Figure 1: (a) is the entropy field of the five critical points data set. (b) shows the critical regions identified from the entropy field. Different colors are for different regions. (c) shows skeleton lines extracted from critical regions.

view path is constructed by interpolating a B-spline curve traversing selected viewpoints in order. For all critical regions in the field, a global B-spline curve path traversing all these regions is generated by picking the one that has the minimal *traversal cost* for all critical regions.

4 Critical Region Identification and Skeleton-based Seeding

4.1 Entropy Field Computation

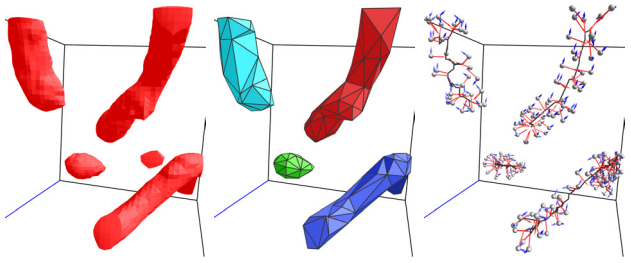
Given an input 3D flow field, we first compute its corresponding scalar entropy field. We employ a $9 \times 9 \times 9$ cube centered at each voxel and compute its entropy for this local region by evaluating the variation of both *direction* and *magnitude* of vectors for all voxels within. By applying this process to each voxel in the flow field, we generate an entropy field as shown in Figure 1 (a). We implement entropy computation in the GPU using CUDA. For a data set which cannot be loaded into graphics memory once, we divide it into blocks and compute the entropy field in an out-of-core manner.

4.2 Critical Region Detection

With the entropy field derived, we define *critical regions* in the volume as local neighborhoods in which all the voxels' entropy values are greater than a given threshold. Intuitively, a critical region is a subvolume in the flow field which contains rich information compared with the remaining non-critical ones.

Region size computation Since the shape of a critical region may not be regular, its size also varies dramatically. In our algorithm, we do not consider the regions with fairly small volume size and they are filtered out from the *critical region set R*. To compute the size of a critical region, we apply a region growing algorithm which approximates the region's volume at the voxel level. As we extract connected voxels with their entropy values greater than the given threshold δ_e, the size of the critical region is defined as the number of marked voxels. We apply the same process to each connected region until we compute the sizes for all critical regions. An example is shown in Figure 1 (b).

Region skeleton extraction In order to identify the shape pattern for each region r in R, we extract its *skeleton* by adopting a volume thinning algorithm developed by Gagvani and Silver [3]. The algorithm computes the *distance transform* which is the distance from the internal voxel of the region to the boundary voxel. Skeleton points are those whose distance transform values are larger than a given *thinness parameter* δ_t. By controlling the value of δ_t, the algorithm determines the density of the final skeleton points. Furthermore, by applying a *minimum spanning tree* (MST) algorithm to the skeleton points, we can eventually connect all skeleton points to form a tree-structured skeleton line, as shown in Figure 1 (c). We identify two endpoints on the skeleton which have the longest Euclidean distance and define the *major direction* of the skeleton as a vector starting from one endpoint of the skeleton with lower y value to the other one. Skeleton extraction plays an important role in our algorithm since it not only indicates the shape pattern but also provides a central curve to focus on for all viewpoints associated with the critical region. In fact, the viewing centers of viewpoints for a critical region will always be positioned on its skeleton.



(a) (b) (c)

Figure 2: (a) shows the isosurfaces constructed for the five critical points data set. Each closed surface indicates one critical region in the flow field. (b) shows the simplified triangle meshes constructed from the isosurfaces in (a) where a region with small volume size (smaller than $1/1000$ of the total flow field volume) is filtered out in advance. (c) indicates all viewpoints generated on the simplified mesh surface. Spheres indicate viewpoint positions while red and blue arrows indicate `look-at` and `up` directions, respectively.

4.3 Skeleton-based Seeding

Seed placement is a central issue in streamline visualization. In order to well cover all critical regions, we adopt a skeleton-based seeding strategy by always dropping seeds along the skeleton of each critical region. This strategy also helps reduce redundant streamlines in uninteresting regions. Since all seeds are aligned along the skeleton with an equal separating distance, we can easily control the density of streamlines by adjusting the distance between two adjacent seeds. Figure 5 (a) shows one example of the streamlines generated using skeleton-based seeding.

5 Viewpoint Creation and Quality Evaluation

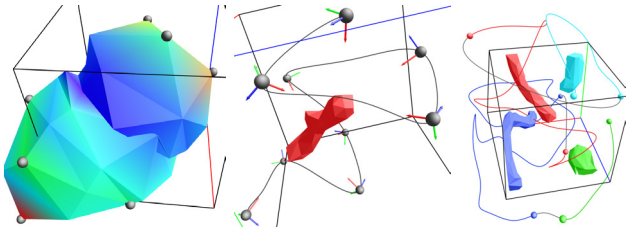
5.1 Isosurface Construction

We leverage the isosurface to indicate the shape and location of each critical region and define the isovalue as the given entropy threshold δ_e (Section 4.2). To obtain the isosurface, we use the classical marching cube algorithm [9]. Figure 2 (a) shows an example of the constructed isosurfaces. Similar to entropy computation, for a large input data set, we leverage CUDA to extract the isosurfaces block by block in the GPU. We also utilize a k-d tree data structure to store all the resulting triangles for fast access. The marching cube algorithm only produces isolated triangles and no geometric connectivity information is readily available for use. Therefore, we convert the isosurfaces into triangle meshes.

During viewpoint creation, we leverage the vertices on each triangle mesh to obtain the viewpoints for each critical region. However, since the isosurface construction is processed in the voxel level, there may be more than thousands of vertices in each mesh. To reduce the number of vertices to a manageable level, we apply a mesh decimation algorithm based on edge collapse introduced by Hoppe [4]. A *decimation factor* δ_s is provided to users for controlling the simplification level of the final mesh. For our application, it is desirable to keep the final number of vertices on the simplified mesh surface to a few hundred. Figure 2 (b) shows an example of the simplified meshes.

5.2 Viewpoint Creation

Given the critical region set R obtained in the first stage, our algorithm creates a list of viewpoints. First, by locating the viewpoints at the vertices on the simplified mesh surface, we obtain a set of viewpoints S. For each viewpoint in S, we compute its `look-at`



(a) (b) (c)

Figure 3: (a) shows an offset surface of one critical region of the five critical points data set where color mapping indicates viewpoint quality. Each black point indicates one best viewpoint. (b) shows the B-spline curve path that connects selected viewpoints in (a). The red, green and blue arrows indicate the `look-at`, `up` and binormal directions of each viewpoint, respectively. (c) shows the final global B-spline curve path traversing all critical regions.

(i.e., viewing) center and `up` direction. For the `look-at` center, we compute the distance between the viewpoint location and the corresponding skeleton line. The point on the skeleton line closest to the viewpoint location is the `look-at` center for this viewpoint. This treatment guarantees that our viewpoint always focuses on the portion of critical region closest to it. The `look-at` direction $\mathbf{l}$ is determined as the vector from the viewpoint location to the `look-at` center. One simple way to compute the `up` direction is to fix it to a predefined direction such as the positive y direction of the volume. However, it does not consider the major direction of the underlying flow patterns and could cause the user to lose the focus and context, especially when the flow direction changes frequently. Therefore, we utilize the skeleton's major direction as a guide to compute the `up` direction. Specifically, we define the local skeleton direction $\mathbf{d}$ at the `look-at` center as the vector along the skeleton which starts from the `look-at` center and points toward the skeleton's major direction. We then project $\mathbf{d}$ onto a plane perpendicular to $\mathbf{l}$. The final `up` direction is the projected vector on the plane. In Figure 2 (c), we show all the viewpoints generated at the vertices on different surfaces. The corresponding `look-at` directions and `up` directions are also displayed.

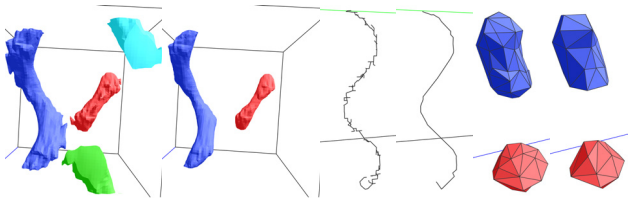
Once we finish computing each viewpoint v in S, we also generate several *offset viewpoints* associated with v by offsetting v along the opposite direction of its `look-at` direction $\mathbf{l}$ for some levels. Offset viewpoints share the same `look-at` center and `up` direction with v and their locations are simply pushed away from v. Intuitively, each offset viewpoint of v is a zoom-out view. We define the offset viewpoints along with v as a *viewpoint set* V. For each viewpoint in V, we evaluate its quality and then select one viewpoint with the highest quality as the representative of V. This procedure is applied to all viewpoint sets on the mesh surface. We then connect the representatives by following the original mesh connectivity information to form a new *offset surface*. Figure 3 (a) shows such an offset surface.

5.3 Quality Evaluation

For each critical region r, we evaluate the quality of viewpoints associated with it based on the amount of information revealed from streamlines seeded from its skeleton. We also consider foreground streamline occlusion and background streamline noise as penalties to avoid visual clutter and distraction. Specifically, we utilize the *mutual information* (MI) between 3D streamline and its 2D projection as the measure of information revealed [16]. We compute the final *viewpoint quality* for a viewpoint v as follows

$$Q(v) = S_{\text{focus}} - (P_{\text{fore}} + P_{\text{back}}), \tag{1}$$

where S_{focus}, P_{fore}, P_{back} are *focus region score*, *foreground occlusion penalty* and *background noise penalty*, respectively.



(a) (b) (c) (d) (e) (f)

Figure 4: Parameter influence. (a) and (b) show critical regions detected for the ABC flow data set with two different δ_e values. (c) and (d) depict the skeleton of a critical region for the tornado data set with two different δ_t values. (e) and (f) show the simplified meshes for the electron data set with two different δ_s values.

Focus region score S_{focus} indicates how much information revealed by the streamlines seeded from the corresponding critical region is preserved under v. To obtain this value, we first check if a streamline s is seeded from the corresponding critical region r. If it is true, we then perform a viewing frustum culling operation to s in order to determine which segments of s are inside of the viewing frustum. For those segments inside, we accumulate the MI of each point along the segments and define the final aggregated MI values as S_{focus}.

Foreground occlusion/background noise penalty Since our viewpoints are mostly located inside of the flow field, it is inevitable that some streamlines seeded from other critical regions will block our view when we look at the critical region r in focus from a given viewpoint v. Additionally, for streamlines behind r, they would potentially distract our attention by adding some "noise" in the final image. We quantize these two effects by defining P_{fore} and P_{back} and select viewpoints with low values from these two terms. We compute these two terms in a single phase. First, we transform the standard OpenGL view projection plane with dimension of 2×2 to a predefined $n \times n$ projection plane $\mathbf{P}$. We then record the minimum Z value for each pixel in $\mathbf{P}$ covered by streamline segments computed in S_{focus}. For pixels not covered, we set an infinitesimal value for them. Next, for each streamline s seeded out of r, we check the Z value for every point along s inside of the viewing frustum and compare it with the Z value of the corresponding pixel in $\mathbf{P}$. If the new value is larger than the one in $\mathbf{P}$, we set that value to $\mathbf{P}$ and mark the point as an occlusion or noise point. If the point is between the viewpoint v and our critical region r in focus, it is an *occlusion point*. If it is at the back of r, it is a *noise point*. P_{fore} and P_{back} are obtained as the summation of the MI values of these two kinds of points, respectively.

6 Viewpoint Selection and Tour Generation

6.1 Best Viewpoints Selection

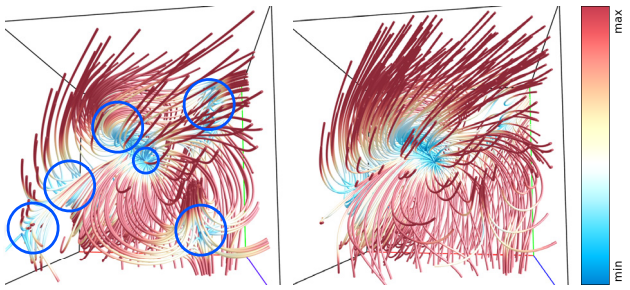
At the end of viewpoint creation step, we construct a new offset surface by connecting all representative viewpoints from each viewpoint set V. Next, we sort all these representatives by their quality and pick the final best viewpoints with the highest values. However, if we take the quality value as the only criterion for best viewpoints selection, neighboring viewpoints with similar high values will be selected together. To avoid this, we define the distance between two viewpoints as a vector with two components: the angle between their `look-at` directions and the distance between their `look-at` centers. Given two distance vectors $\mathbf{d}_1$ and $\mathbf{d}_2$, we then define $\mathbf{d}_1 > \mathbf{d}_2$ when either one of the two components in $\mathbf{d}_1$ is greater than the corresponding one in $\mathbf{d}_2$. Leveraging this measure, we require that the distance between any two best viewpoints selected should be greater than a given distance threshold δ_d. In Figure 3 (a), we depict how the best viewpoints are arranged on an offset surface.

data set	dimension	entropy field computation	critical region detection	initial #lines	isosurface construction	mesh conversion	viewpoint creation	#best viewpoints/ #total viewpoints	viewpoint evaluation	tour path generation
five critical pts	$51 \times 51 \times 51$	0.473s	0.119s	800	0.085s	0.139s	0.015s	27/175	106.073s	0.084s
two swirls	$64 \times 64 \times 64$	0.507s	0.158s	500	0.161s	0.399s	0.027s	25/239	218.293s	0.074s
tornado	$64 \times 64 \times 64$	0.543s	0.223s	400	0.165s	0.179s	0.031s	12/244	98.948s	0.019s
supernova	$200 \times 200 \times 200$	7.891s*	570.826s	400	3.953s*	4.334s	0.135s	10/766	266.470s	0.013s
solar plume	$126 \times 126 \times 512$	8.255s*	494.908s	400	4.073s*	7.748s	0.129s	15/751	381.420s	0.021s
ABC flow	$64 \times 64 \times 64$	0.610s	0.125s	1000	0.173s	0.152s	0.014s	12/157	65.092s	0.091s
electron	$64 \times 64 \times 64$	0.560s	0.062s	450	0.159s	0.055s	0.008s	7/68	10.753s	0.011s

Table 1: The timing results for seven flow data sets. A * denotes out-of-core processing in the GPU using CUDA.

data set	entropy δ_e	thickness δ_t	decimation δ_s	distance δ_d	angle δ_α
five critical pts	2.845	0.7	0.320	10.0	$\pi/4$
two swirls	2.930	0.7	0.328	10.0	$\pi/4$
tornado	2.098	0.7	0.328	10.0	$\pi/6$
supernova	2.459	0.9	0.335	15.0	$\pi/3$
solar plume	2.939	0.9	0.335	12.0	$\pi/3$
ABC flow	2.306	0.7	0.320	10.0	$\pi/4$
electron	2.092	0.7	0.320	10.0	$\pi/4$

Table 2: The parameter settings for seven flow data sets.



(a) (b)

Figure 5: Comparing (a) skeleton-based seeding and (b) random seeding for the five critical points data set. Both place 320 streamlines. Velocity magnitude is mapped to streamline color.

6.2 Tour Path Generation

Straight path generation From a set of best viewpoints selected, we need to form a path traversing all of them. In order to guarantee that the path follows the skeleton's shape pattern, we utilize the skeleton's major direction as a guide for path generation. Specifically, we pick the viewpoint whose center is closest to one end of the skeleton's major direction as the starting point of the view path and set it as a *pivot viewpoint* v_p. We then connect the viewpoint v whose `look-at` center is closest to the current v_p's `look-at` center (i.e., $d_c(v_p, v)$ is the smallest) and set it as the new v_p. To avoid the zig-zag path shape, we force the angle formed by every three consecutive viewpoints along the path to be larger than a given threshold δ_α. After we identify the first two viewpoints v_1 and v_2 along the path, we identify the newly selected viewpoint v_3 so that $\angle v_1 v_2 v_3$ is larger than δ_α and at the same time, $d_c(v_p, v_3)$ is small enough. To achieve this, we first sort the remaining viewpoints based on their `look-at` center distances to v_p, then check the angle formed by v_1, v_2, and each of the remaining viewpoints. The first viewpoint whose angle is larger than δ_α is selected and set to be the new v_p. We repeat this process until the whole path has been generated. Finally, we create a straight view path by connecting all viewpoints using line segments. One drawback of using the straight-line as the view path is that in some cases, line segments may intersect with the skeleton (i.e., the view path will get fairly close to the flow feature). To avoid this, we replace the straight-line with a B-spline curve to "bend" the path away from the skeleton when they are too close to each other.

B-spline curve path generation In order to keep the view path away from the corresponding critical regions skeleton for some distance, we add one intermediate viewpoint between each pair of adjacent viewpoints along the straight path. Essentially, for each line segment along the straight path, we first compute the shortest distance $d_{l,s}$ between the segment and the skeleton. Assuming p_l and p_s are two points on the line segment and the skeleton respectively and their distance is $d_{l,s}$, we push p_l away from the skeleton along the direction of vector $\overrightarrow{p_s p_l}$ for some distance Δ_d, where Δ_d is inversely proportional to $d_{l,s}$. Intuitively, p_l will be pushed far away from the skeleton (i.e., larger Δ_d) if its corresponding line segment is close to the skeleton (i.e., small $d_{l,s}$). After creating all intermediate viewpoints, a B-spline curve path which traverses all viewpoints including intermediate ones is interpolated. By setting all viewpoints as data points $\mathbf{b}$ on the curve, we can compute the B-spline basic functions for each data point and then define a matrix $\mathbf{N}$ by setting all the functions of each data point as a single row. Given $\mathbf{N}$ and $\mathbf{b}$, we interpolate the B-spline curve by solving the following linear system

$$\mathbf{N}\mathbf{x} = \mathbf{b}, \tag{2}$$

where $\mathbf{x}$ represents the control points of the B-spline. To avoid getting a singular linear system, we generate knots by averaging the parameters. We then generate the viewpoints on the curve by utilizing $\mathbf{x}$ and their corresponding basic function values. We always use a B-spline of degree three as our final path representation. Figure 3 (b) shows an example of the final curve path. In order to maintain a smooth animation when moving viewpoints along the path, we also reparameterize the path and interpolate several viewpoints between every two adjacent best viewpoints with equal arc length.

From single region to multiple regions The preceding algorithm operates on a single region. If there is more than one critical region in the field, a global B-spline curve path traversing all these regions is generated. To achieve this, we first compute and sort the best viewpoints for each critical region as we describe in the straight path generation phase. Next, we order all the regions so that the final global path could traverse all the best viewpoints in a smooth and efficient way. Specifically, we define the *traversal cost* $C(r_1, r_2)$ from region r_1 to r_2 by considering two factors: the distance $D(r_1, r_2)$ between the positions of the last viewpoint of r_1 and the first viewpoint of r_2, and the angle $A(r_1, r_2)$ between the `look-at` directions of these two end viewpoints. That is,

$$C(r_1, r_2) = D(r_1, r_2) * A(r_1, r_2), \tag{3}$$

where both distance and angle are normalized by their corresponding maximum values. Intuitively, we connect two regions if their end viewpoints are spatially close to each other and their angle change is small. The cost of the global path is the summation of the traversal cost between each pair of critical regions traversed in order. We compute the costs for all possible region orderings, i.e., for each critical region, we also take into account the two different orderings of its end viewpoints. From all these orderings, we select the final global path as the one with the smallest cost value. For efficiency, we prune those orderings if we encounter excessively large

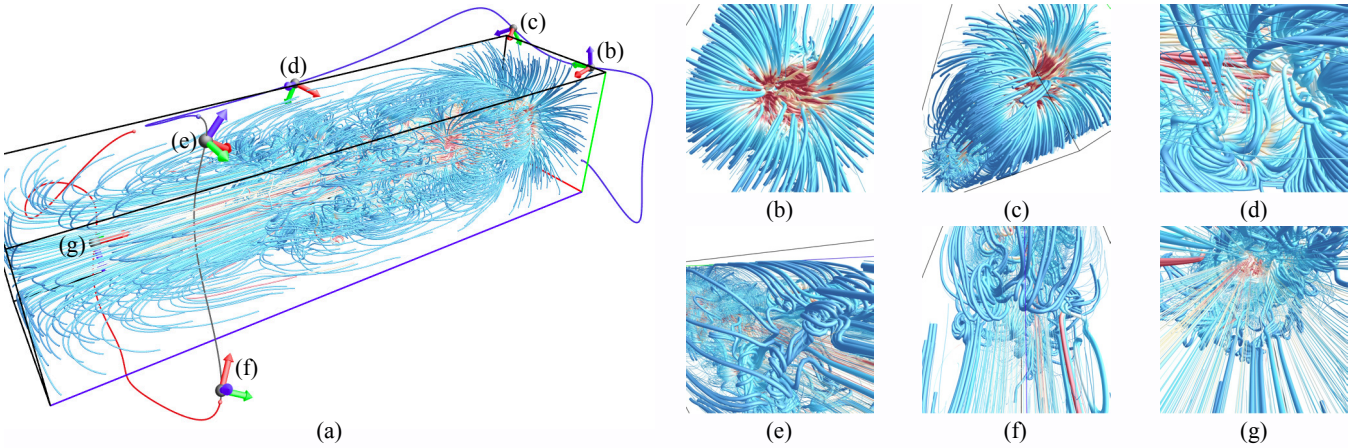


Figure 6: FlowTour screenshots for the solar plume data set. (b) to (g) show the respective views from six different viewpoints along the tour path as marked in (a). Tube radius is decreased in (a) for all streamlines so that the view path can be perceived.

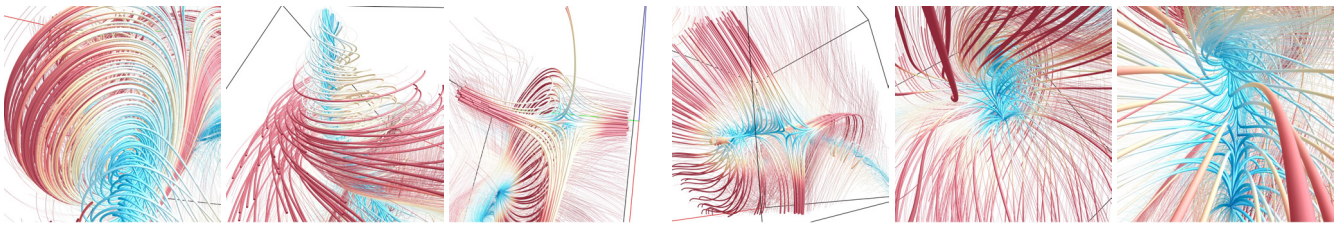


Figure 7: FlowTour screenshots for the five critical points data set. The five critical points are two spirals (a) and (b), two saddles (c) and (d), and a source (e). (f) shows the connection between a spiral and the source.

$D(r_1, r_2)$ or $A(r_1, r_2)$ values along the traversal. Finally, a global B-spline curve is interpolated by considering all the viewpoints on the global path as data points. Figure 3 (c) shows an example of the global path.

6.3 Viewpoint Traversal and Path Animation

Given the final global B-spline tour path, we traverse all viewpoints for each critical region in order by moving the camera along the path. Whenever there is an abrupt change of viewing angles between two adjacent viewpoints, we interpolate intermediate viewpoints for a smooth transition. We render streamlines as tubes. To help users focus on the currently traversed region, we render the streamlines seeded from the current focus region with a large tube radius and all other streamlines with a small tube radius. When the camera focus changes from one critical region to another, an animated transition indicating the changes of streamline thickness is shown. In the user study, we also provide users with the freedom to change the animation speed, pause the animation, or play the animation in reverse order so that they can observe critical regions in a more flexible manner.

7 Results and Discussion

In this section, we report the performance, parameter, and image results gathered from several flow data sets. To best evaluate our approach, please refer to the accompanying video which shows the animation of FlowTour.

7.1 Configurations and Timing

We implemented FlowTour on a CPU-GPU hybrid platform 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. Entropy field computation, isosurface extraction, and viewpoint quality evaluation were implemented in the GPU using CUDA and all other computations were implemented in the CPU. Since we changed streamline thickness frequently, we utilized the *vertex buffer object* (VBO) to render streamlines and used the GPU to process their geometry changes in order to provide smooth streamline update. The timing results and parameter settings for the seven data sets we used are reported in Tables 1 and 2. All stages of processing can be finished within 15 minutes for each data set.

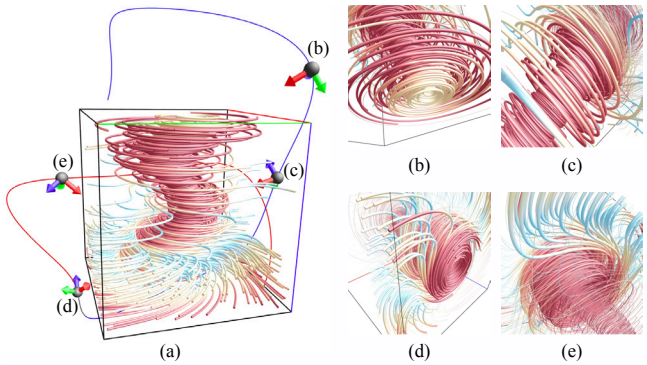
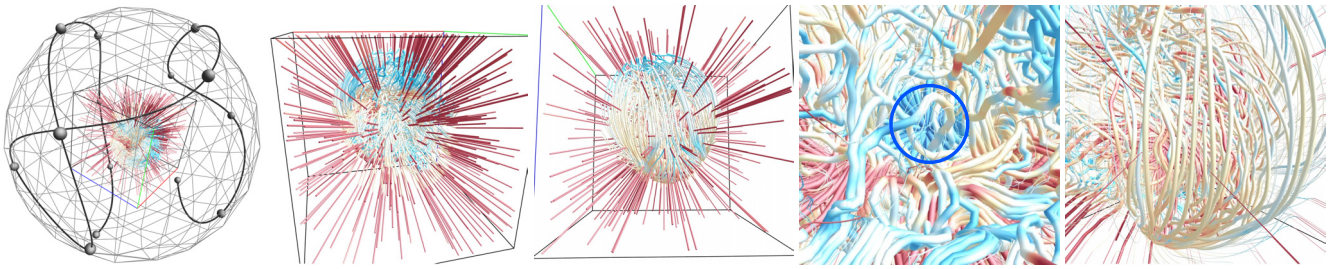


Figure 8: FlowTour screenshots for the tornado data set. (b) to (e) show the respective views from four different viewpoints along the tour path as marked in (a).

7.2 Parameter Influence

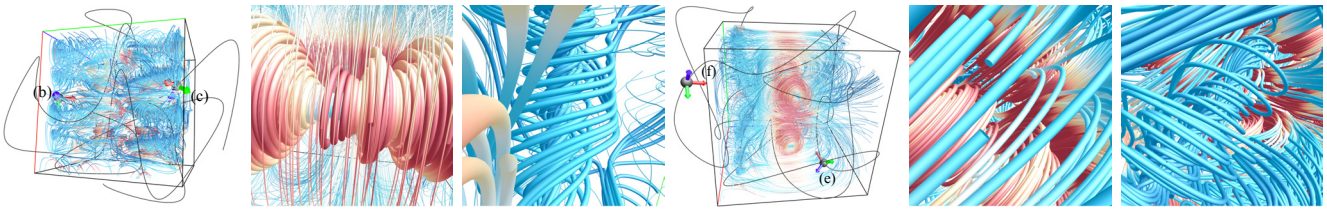
Parameters play an important role in our algorithm as users can change their values to adjust the results. Here we discuss how three parameters could affect the results for several data sets.

Entropy threshold δ_e Figure 4 (a) and (b) show the results of critical regions detected with different entropy threshold δ_e values



(a) (b) (c) (d) (e)

Figure 9: FlowTour vs. external tour for the supernova data set. (a) shows the bounding sphere and the selected best viewpoints for external tour. (b) and (c) are screenshots corresponding to two different viewpoints for external tour. (d) and (e) are screenshots from FlowTour.



(a) (b) (c) (d) (e) (f)

Figure 10: FlowTour vs. random tour for the two swirls data set. (a) and (d) show the tour paths for FlowTour and random tour, respectively. (b) and (c) are screenshots from FlowTour, while (e) and (f) are screenshots from random tour. The corresponding two viewpoints for the two methods are marked in (a) and (d), respectively. Tube radius is decreased in (a) and (d) for all streamlines so that the view path can be perceived.

used for the ABC flow data set. As we can see, this parameter provides users with the freedom to select the final critical regions at a different size level. Furthermore, since δ_e is the isovalue for determining the isosurface in the flow field, changing δ_e helps users locate the critical regions at different scales.

Thickness threshold δ_t In Figure 4 (c) and (d), we show the results of the detected critical regions and their corresponding skeletons with various thickness threshold δ_t values used for the tornado data set. Changing this parameter allows users to control the density of skeleton points as well as the shape of the skeleton curve which is a MST of skeleton points. Since both the `look-at` and `up` directions of our best viewpoints are computed based on the region skeleton, users can construct various tour paths by choosing different δ_t values.

Decimation factor δ_s The decimation factor provides users with the freedom to control the number of vertices on the simplified mesh. Since our initial viewpoint pool is generated from these vertices, changing δ_s allows users to build their own initial viewpoint pool which also affects the final selected best viewpoints. Figure 4 (e) and (f) show two different simplified meshes for the same critical regions with different δ_s values for the electron data set.

7.3 Skeleton-based Seeding vs. Random Seeding

In Figure 5, we compare our skeleton-based streamline seeding with random seeding. Clearly, our method can convey more information of the original flow field than random seeding since most of the streamlines in our method are located around the critical regions (highlighted with circles in Figure 5 (a)) which are the most interesting areas of the flow field. Furthermore, unlike random seeding which places streamlines arbitrarily in the field, our method also reduces streamline occlusion since we never put seeds in uninteresting regions. This would help users observe flow field patterns in a less ambiguous way.

7.4 FlowTour Exploration

Figure 6 shows the final B-spline tour path and six screenshots along the FlowTour exploration of the solar plume data set. As we can see in (a), the curve tour provides a smooth traversing path which effectively covers most features of the flow field. (c) depicts the major flow pattern, i.e., the head of the solar plume, which provides users with a good overall view of the flow field. This was made possible with our offset viewpoints which make sure that the viewpoints are not too close to the scene. (b) shows the zoom-in effect of the head of the solar plume. The "flower"-like pattern is clearly depicted and the velocity variation around the core of the head is also nicely revealed. In (d), some small spiral patterns inside of the head of the solar plume are shown. Since there are many streamlines around this region, it is difficult for users to observe such features from external views. In (e) and (f), some detailed patterns such as small spirals around the straight lines in the middle portion of the flow field are also clearly captured from two different viewpoints. Instead of always forcing users to look at the flow field from outside, FlowTour can also take users to the "kernel" of the flow field and provide an expressive traversal experience which is not available using external view alone. (g) gives such an example. Users can now clearly observe the flow patterns of the internal hollow shaft by "standing" right inside of it.

In Figure 7, we show the FlowTour results for the five critical points data set. The five critical points are clearly depicted from (a) to (e). Furthermore, our tour also captures the connection between the source and a spiral as shown in (f). Since this connection is in the center of the flow field and is occluded by many surrounding streamlines, it is more difficult to detect such a connection if we only use external views.

For the tornado data set shown in Figure 8 (b) and (d), two distinct patterns at the two ends, i.e., the "S" shape on the bottom of tornado and the large cap on the top of tornado, are clearly depicted. (c) and (e) show two side views. In (a), we show the tour which smoothly traverses the two major critical patterns of the flow field.

7.5 FlowTour vs. External View Tour

We compare FlowTour with a traditional external view tour. Both use the same set of streamlines placed using our skeleton-based seeding. As shown in Figure 9 (a), for the external view tour, all viewpoints are uniformly assigned on a bounding sphere which encloses the flow field. The `look-at` focus of each viewpoint is the

center of the flow field and the up direction is always along the x, y or z axis. The viewpoint score is computed as the summation of mutual information values for each streamline under this viewpoint. We select the best viewpoints by following the same strategy we use for FlowTour: sorting the viewpoints by their scores and picking best viewpoints according to their distance to the viewpoints which have been picked already. We then interpolate the final B-spline path going through all best viewpoints and ensure that the path always stays outside of the flow field. Figure 9 (b) and (c) show two different screenshots for external view tour. It is obvious that although the external view tour could depict the overall flow field clearly, the detailed patterns, especially around the core of the supernova, are largely occluded by surrounding streamlines. Conversely, FlowTour not only captures the overall shape of the flow field, as shown in (e), but also reveals the hidden source in the center of the supernova, as marked in (d).

7.6 FlowTour vs. Random View Tour

We also compare FlowTour with a naïve random view tour. In order to avoid bias, we keep both the path length and the total angle change the same for these two methods. For the random method, we first generate a set of viewpoints whose positions are randomly picked. Similar to FlowTour, we allow some viewpoints to be outside of the flow field boundary in order to gain a global overview of the data set. The look-at direction and up direction of each viewpoint are also randomly created. Next, we pick the viewpoint with the smallest x, y and z coordinates as the starting point of the path and connect other viewpoints by following the same strategy used in FlowTour, which minimizes both the Euclidean distance between viewpoints and the angle change along the path. Finally, we interpolate a new B-spline curve passing all the viewpoints to generate the path. We show a comparison between these two methods in Figure 10. For FlowTour, we can see from (b) and (c) that the internal swirling patterns and small spiral features are clearly shown in great detail. Specifically, (b) indicates that the radii of the major swirls inside of the volume are not always the same. They actually vary along the axle, which is not captured from any external viewpoint. In (c), the small boundary spiral patterns are shown clearly. For the random view tour, (e) and (f) show the same flow patterns depicted in (b) and (c), respectively. However, since the viewpoints are not selected intentionally for observing any flow patterns, they cannot guarantee clear exploration of these internal features. Actually, the random view tour not only performs poorly on exploring flow features, but also fails to provide a smooth viewpoint change along the view path (refer to the accompanying video).

8 User Study

We conducted a user study to evaluate the effectiveness of our FlowTour. We used a design of 3 conditions (FlowTour path, randomly-generated internal path, and external path) $\times$ 2 tasks (answer questions and identify critical regions). The random internal path was constrained to be the same length with the same amount of total rotation as the FlowTour path. The external path was calculated using a strategy similar to FlowTour, with skeleton-based seeding, but constrained to remain outside of the data set. We recruited 10 users for the FlowTour condition, 11 users for the random condition, and 10 users for the external condition. Participants had the option of being paid $10 for participating in the study. All participants were recruited from the university campus and the local community.

For each experimental session, users were shown seven flow field data sets, one for practice and six for evaluation. After each tour, the users were asked several multiple-choice questions about features in the flow field, and were then asked to identify as many critical regions as they could find within a set time limit. We hypothesized that users would perform better on both tasks with FlowTour.

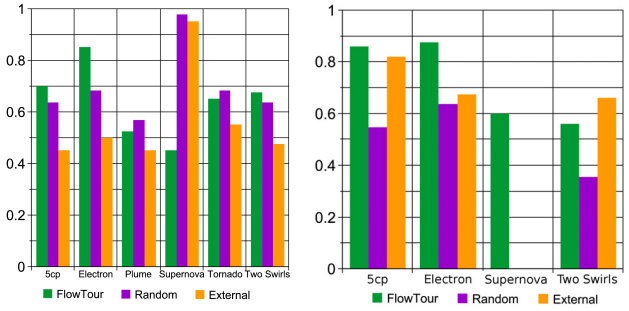


Figure 11: (a) Average proportion of correct answers of multiple-choice questions for each data set. (b) Average proportion of critical regions identified correctly for each data set. For the supernova data set, the score with the random and external paths is zero.

8.1 Experimental Procedure

First, users were given a briefing explaining the basic characteristics of flow fields, critical regions they would need to recognize, and the tasks they would be expected to perform. After the briefing, the users were given the chance to practice their tasks on one data set before beginning the main study. The practice session took the users through one complete set of tasks and gave them the opportunity to become familiar with the program interface and working with flow fields. After the users finished the practice set, the main study commenced, consisting of six data sets. At this time, users were only allowed to ask the researchers for clarification about the meaning of specific questions or how to use the program.

The procedure for each data set was as follows. The user was shown an animation of the complete path through the data set. The speed of the animation could be adjusted if desired. After the animation, the user was presented with several questions and then asked to identify critical regions in the data set. The user had a limited time to perform these tasks. While performing these tasks, the user could revisit any part of the path using a slider. This functionality was useful for answering questions and was required for identifying critical regions.

Users completed all seven data sets in one sitting. They were not allowed to take breaks or leave the workstation until the experiment was completed. Users could opt to terminate the experiment at any time, but none chose to do so. The entire experiment took approximately 60 minutes for most users, including initial paperwork, briefing, and post-experiment questionnaire. The questionnaire asked the users for subjective feedback and suggestions for improvement regarding the experiment and the program's user interface.

8.2 Results and Discussion

We present the results of this study in two aspects: user accuracy on multiple-choice questions and the proportion of critical regions correctly identified. Because user performance varied widely by data set, each data set was analyzed individually, comparing user performance by path (FlowTour, random internal, and external). We used one-way ANOVA to analyze statistical significance between the conditions with a standard significance level $\alpha = 0.05$.

Multiple-choice questions Each data set was analyzed individually by comparing users' average proportion of correct answers by different paths. The results are given in Figure 11 (a). Performance differences in this condition were not statistically significant except for the supernova data set, $p < 0.0001$, wherein the random and external paths substantially outperformed the FlowTour path.

Critical regions identification Similar to the multiple-choice questions, each data set was analyzed individually. The analysis was done by comparing how many critical regions the user correctly identified against the total number of critical regions in the data set, such that a value of 1.0 indicates the user found every critical region. The results are given in Figure 11 (b). The solar plume and tornado data sets were excluded from this analysis, the former because it is a turbulent mass that lacks coherent critical regions, the latter because it contains one enormous critical region that is easily identifiable by all users in all conditions. In all cases, users correctly identified more critical regions with the FlowTour path, except for the two swirls data set where the external path slightly outperformed the FlowTour path. All of these results are statistically significant ($p < 0.05$).

Discussion The overall result of the experiment is that, in the multiple-choice questions condition, users performed better with the FlowTour path and the random internal path over the external path, albeit without statistical significance in most cases. Conversely, on most data sets, users performed better at identifying critical regions with the FlowTour path. One exception to this pattern is that users answered questions about the supernova data set more accurately with the random and external paths. This might be due to the fact that the supernova contains only a single critical region surrounded by chaotic turbulence, which hinders the FlowTour algorithm from plotting an effective path through the data set.

The multiple-choice questions were designed to evaluate users' overall knowledge of the general characteristics of each data set. Lack of statistical significance hinders us from drawing definitive conclusions, but the fact that both the FlowTour path and the random internal path outperformed the external path in all but one case suggests that being able to view the interior of a data set may indeed provide users with better overall knowledge of that data set. Additionally, the relatively good performance of the random path in this task suggests that getting a general "feel" for a data set may not require a sophisticated approach. Conversely, users performed better at identifying critical regions with the FlowTour path. This result is not surprising since exposing critical regions is a major objective of the FlowTour algorithm. These results demonstrate that finding and recognizing critical regions—and by extension, other kinds of specific flow patterns—is enhanced by an algorithmic approach designed to aid this task, and that a random approach is not adequate to ensure optimal performance.

We conclude that FlowTour is not significantly more useful than a haphazard approach when gathering general information about flow fields, but FlowTour is effective in aiding users to identify critical regions, especially hidden or occluded flow features.

9 Concluding Remarks

We have presented FlowTour, a new solution to explore internal flow features and patterns. Many of these internal views reveal novel flow information which is difficult to acquire using external views only. Our solution thus nicely complements external view exploration by providing insightful viewpoints for observing complex flow fields with hidden or occluded features. As the size and complexity of flow field data keep growing, there is an increasing need to allow users to observe flow features and patterns from inside the volume. Therefore, the general approach presented in this paper would become more important in various visual analysis tasks and we expect that future flow visualization systems will be equipped with functions comparable to FlowTour.

Acknowledgements

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

References

[1] U. D. Bordoloi and H.-W. Shen. View selection for volume rendering. In *Proceedings of IEEE Visualization Conference*, pages 487–494, 2005.

[2] M. Feixas, M. Sbert, and F. González. A unified information-theoretic framework for viewpoint selection and mesh saliency. *ACM Transactions on Applied Perception*, 6(1), 2009.

[3] N. Gagvani and D. Silver. Parameter-controlled volume thinning. *Graphical Models and Image Processing*, 61(3):149–164, 1999.

[4] H. Hoppe. Progressive meshes. In *Proceedings of ACM SIGGRAPH Conference*, pages 99–108, 1996.

[5] G. Ji and H.-W. Shen. Dynamic view selection for time-varying volumes. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1109–1116, 2006.

[6] B. Jobard and W. Lefer. Creating evenly-spaced streamlines of arbitrary density. In *Proceedings of the Eurographics Workshop on Visualization in Scientific Computing*, pages 43–56, 1997.

[7] T.-Y. Lee, O. Mishchenko, H.-W. Shen, and R. Crawfis. View point evaluation and streamline filtering for flow visualization. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 83–90, 2011.

[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] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of ACM SIGGRAPH Conference*, pages 163–169, 1987.

[10] J. Ma, C. Wang, and C.-K. Shene. Coherent view-dependent streamline selection for importance-driven flow visualization. In *Proceedings of IS&T/SPIE Conference on Visualization and Data Analysis*, 2013.

[11] S. Marchesin, C.-K. Chen, C. Ho, and K.-L. Ma. View-dependent streamlines for 3D vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1578–1586, 2010.

[12] A. Mebarki, P. Alliez, and O. Devillers. Farthest point seeding for efficient placement of streamlines. In *Proceedings of IEEE Visualization Conference*, pages 479–486, 2005.

[13] O. Rosanwo, C. Petz, S. Prohaska, H.-C. Hege, and I. Hotz. Dual streamline seeding. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 9–16, 2009.

[14] B. Spencer, R. S. Laramee, G. Chen, and E. Zhang. Evenly spaced streamlines for surfaces. *Computer Graphics Forum*, 28(6):1618–1631, 2009.

[15] S. Takahashi, I. Fujishiro, Y. Takeshima, and T. Nishita. A feature-driven approach to locating optimal viewpoints for volume visualization. In *Proceedings of IEEE Visualization Conference*, pages 495–502, 2005.

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

[17] G. Turk and D. Banks. Image-guided streamline placement. In *Proceedings of ACM SIGGRAPH Conference*, pages 453–460, 1996.

[18] P.-P. Vázquez, M. Feixas, M. Sbert, and W. Heidrich. Viewpoint selection using viewpoint entropy. In *Proceedings of Vision, Modeling, and Visualization Conference*, pages 273–280, 2001.

[19] V. Verma, D. Kao, and A. Pang. A flow-guided streamline seeding strategy. In *Proceedings of IEEE Visualization Conference*, pages 163–190, 2000.

[20] I. Viola, M. Feixas, M. Sbert, and M. E. Gröller. Importance-driven focus of attention. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):933–940, 2006.

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