Image-Based Streamline Generation and Rendering

Liya Li and Han-Wei Shen

Abstract—Seeding streamlines in 3D flow fields without considering their projections in screen space can produce visually cluttered rendering results. Streamlines will overlap or intersect with each other in the output image, which makes it difficult for the user to perceive the underlying flow structure. This paper presents a method to control the seeding and generation of streamlines in image space to avoid visual cluttering and allow a more flexible exploration of flow fields. In our algorithm, 2D images with depth maps generated by a variety of visualization techniques can be used as input from which seeds are placed and streamlines are generated. The density and rendering styles of streamlines can be flexibly controlled based on various criteria to improve visual clarity. With our image space approach, it is straightforward to implement the level of detail rendering, depth peeling, and stylized rendering of streamlines to allow for more effective visualization of 3D flow fields.

Index Terms—Three-dimensional flow visualization, streamlines, streamline seeding, level of detail, depth peeling, stylized rendering.

1 Introduction

Effective visualization of 3D vector fields plays an important role in many scientific and engineering disciplines such as climate modeling, computational fluid dynamics, and automobile design. To visualize 3D vector fields, many techniques have been developed in the past. Existing techniques include geometry-based methods such as streamline and particle animations, as well as more advanced techniques that utilize graphics hardware to generate realistic flow textures. Generally speaking, texture-based methods such as Line Integral Convolution (LIC), Spot Noise, or the more recent Image-Based Flow Visualization (IBFV) [23] techniques are mostly suitable for visualizing 2D flows. When extending those techniques to 3D data, occlusion becomes a major problem. It is also a challenge to apply the texture-based methods to unstructured grids, although the recent work by van Wijk [24] and Laramee et al. [10] have provided clever solutions to compute flow textures for arbitrary surfaces.

Compared to texture-based approaches, visualizing streamlines is still more widely used because it is easier to compute and render those lines interactively. The main challenge for visualizing streamlines, however, is that the scene can easily become cluttered when too many lines are displayed simultaneously. In addition, it is more difficult to add effective depth cues to points and lines; consequently, understanding the spatial relationships among them can be challenging. Previously, researchers have attempted to develop effective seed placement algorithms for better visualization of streamlines [7], [16], [21], [25]. However, most of the algorithms were developed for 2D vector fields and thus cannot be directly applied to visualizing 3D data.

In this paper, we present an image-based approach for streamline generation and rendering. Our goal is to better display 3D streamlines and reduce visual cluttering in the output images. Visual cluttering occurs because streamlines can arbitrarily intersect or overlap with each other after being projected to the screen, which makes it difficult for the user to perceive the underlying flow structures. In our algorithm, instead of placing streamline seeds in 3D space, we drop seeds on the 2D image plane and then unproject the seeds back to 3D object space before streamline integrations take place. The 3D positions of the seeds can be uniquely determined by the selected image positions, and their depth values can be obtained from an input depth map. By carefully spacing out the streamlines in image space as they are integrated, we can effectively reduce visual cluttering and minimize depth ambiguity caused by overlapping streamlines in the 2D image. We can also achieve a variety of effects such as level of detail (LOD), depth peeling, and stylized rendering to enhance the perception of 3D flow lines. Another advantage of our algorithm is that seed placement and streamline visualization become more tightly coupled with other visualization techniques. As the user is exploring other flow-related variables, when interesting features on the screen are spotted, the seeds can be directly placed on the image without the need to have a separate process to find the 3D seed positions surrounding the features.

The remainder of the paper is organized as follows: First, we briefly review the existing work on seed placement and streamline rendering. Then, we introduce our image-based seed placement and streamline generation algorithm. Finally, we present a variety of ways to utilize our image-based algorithm for better visualization of 3D streamlines.

• *The authors are with the Department of Science and Engineering, The Ohio State University, 395 Dreese Laboratories, 2015 Neil Avenue, Columbus, OH 43210. E-mail: {lil, hwshen}@cse.ohio-state.edu.*

Manuscript received 31 Aug. 2006; revised 25 Oct. 2006; accepted 30 Oct. 2006; published online 8 Jan. 2007.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-0152-0806.
Digital Object Identifier no. 10.1109/TVCG.2007.1009.

TABLE 1
Comparison of the Three Approaches

approach	Mattausch et al. [14]	Ye et al. [26]	Ours
seeding strategy	evenly-spaced	topology-based + Poisson	evenly-spaced
seeding space	object space	object space	image space
candidate seeds	object space distance	on templates	screen space distance

2 Related Work

For 2D flow fields, several techniques are available for generating seeds to reduce visual cluttering. The image-guided streamline placement algorithm proposed in [21] uses an energy function to measure the difference between a low-pass-filtered version of the image and the desired visual density. An iterative process is used to reduce the energy through some predefined operations on the streamlines. This algorithm was extended to curvilinear grid surfaces by Mao et al. [13]. The vectors on a 3D surface are first mapped into the computational space of the curvilinear grid. To address the distortion caused by the uneven grid density, they proposed a new energy function to guide the placement of streamlines in the computational space with the desired local densities. For creating evenly spaced streamlines, Jobard and Lefer [7] proposed controlling the distance between two adjacent streamlines to achieve the desired density. Both the seed selection and the termination of integration are controlled by first measuring the distance to the existing streamlines and then determining the desired actions. Mebarki et al. [16] proposed a 2D streamline seeding algorithm by placing a new streamline at the farthest point away from all existing streamlines. Verma et al. [25] used various templates around critical points to reveal important flow features. A Poisson disk distribution is used to randomly distribute additional seed points in those noncritical regions. This work has been extended to 3D flow fields in [26].

Generating aesthetically pleasing streamlines in 3D flow fields is much more difficult than in 2D fields because the projection process from 3D object space to 2D image space can cause overlaps and intersections, which does not happen for 2D streamlines. Ye et al. [26] presented a strategy for streamline seeding through analyzing the flow topology. Critical points are used to identify the flow regions with important features, and then different seeding templates are used at the vicinity of critical points. Finally, Poisson seeding is used to populate the final empty region. Jobard and Lefer's algorithm [7] was extended to 3D by Mattausch et al. in [14]. Spatial perception of the 3D flow was improved by using depth cueing and halos. They also applied focus + context methods, ROI-driven streamline placing, and spotlights to solve the occlusion problem. Table 1 compares the major differences between the two related works and our approach.

There have been some techniques proposed for rendering 3D flow fields with a better perception of spatial information. Lighting is one of the elements to improve spatial perception. Stalling and Zockler [19] employed a realistic shading model to interactively render a large number of properly illuminated field lines using 2D textures. The algorithm is based on a maximum lighting principle, which gives a good approximation of specular reflection. To improve diffuse reflection, Mallo et al. [12] proposed a view-dependent lighting model based on averaged Phong/Blinn lighting of infinitesimally thin cylindrical tubes. They used a simplified expression of cylinder averaging. To emphasize depth discontinuities, Interrante and Grosch [6] used a visibility-impeding 3D volumetric halo function to highlight the locations and strengths of depth discontinuities, whereas Park et al. [17] proposed a dense geometric flow visualization technique. Multidimensional transfer functions are used to address the occlusion problem inherent to dense volumetric visualization. Previous works such as line-art drawing [5] can also be used to improve the quality of streamline visualization.

3 Overview of Approach and Contributions

The primary goal of our research is to control scene cluttering when visualizing 3D streamlines. For 3D data, addressing the issue of visual cluttering in 3D object space is more challenging since, even if streamlines are well organized in object space, they might still clutter together after being projected to the screen. In this research, we are inspired by how artists draw in real life, as shown in Fig. 1: Strokes are drawn one by one onto the canvas; when some region gets cluttered, fewer strokes are placed, and vice versa. To apply this principle to the flow visualization problem, we place the streamlines based on how they are distributed across the image plane.

Fig. 2 shows the visualization pipeline of our image-based streamline seeding and generation algorithm. The inputs to our algorithm are a vector field and a 2D image with a depth map. The 2D image and depth map can come from the result of rendering vector-field-related properties such as stream surfaces or can be the output of other visualization techniques such as isosurfaces or slicing planes of various scalar variables. Our algorithm will generate streamlines by placing seeds on the image plane.

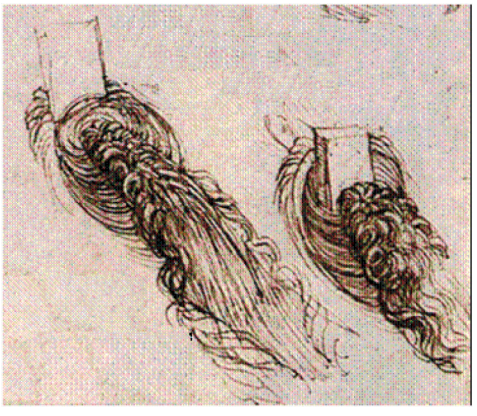


Fig. 1. A drawing of flows by Leonardo da Vinci [2].

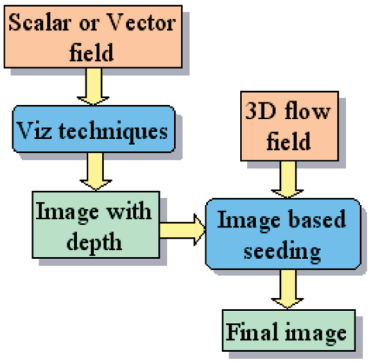


Fig. 2. Visualization pipeline of our image-based streamline generation scheme.

With the depth values of the selected pixels provided by the depth map, those image space seeds can be unprojected back to 3D object space. Streamlines are then integrated in 3D object space. Our algorithm ensures that streamlines will not come too close to each other after they are projected to the screen.

With our algorithm, it is possible to avoid scene cluttering caused by streamlines having a very high depth complexity. Although researchers have previously proposed drawing haloed lines to resolve the ambiguity of streamline depths [14], when a large number of line segments generated from the haloing effect are displayed, the relative depth relationship between the streamlines becomes very difficult to comprehend. By controlling the spacing of streamlines on the image plane, we are able to prevent the visualization from being overly crowded. Another advantage of the image-based approach is that it enhances the understanding of the correlation between the underlying flow field and other scalar variables. When analyzing a flow field, the user often needs to visualize additional variables in order to understand the underlying physical properties in detail. Dropping seeds in the regions of interest defined by those scalar properties can assist in creating a better mental model to comprehend the data. Traditionally, visualization of streamlines and other scalar properties are performed independently. Dropping streamline seeds in regions signified by other data attributes often requires the implementation of the seed placement algorithm to have knowledge about the specific features. In this work, we intend to provide a simple and unified framework by allowing the user to drop seeds directly when they see interesting features in the images. This way, the process of issuing queries to answer the user's hypotheses both in scalar and vector fields can be performed more coherently.

One key issue that needs to be addressed to realize our idea is how to place seeds and generate streamlines so that the visual complexity in the output 2D image is well controlled. We are also interested in exploring a variety of ways to utilize the image space approach to assist us in creating better visualizations of streamlines. In Section 4, we discuss our algorithm in detail.

4 Image Space Streamline Placement

To generate well-organized 3D streamlines, one issue that must be addressed is how to control the spacing between streamlines when they are projected onto the 2D image plane. Previously, researchers have proposed several evenly spaced 2D streamline placement algorithms [7], [16], [21]. Researchers have also extended the idea to 3D vector fields by ensuring evenly spaced streamlines in 3D space [14]. However, such a straightforward extension of 2D streamline placement methods to 3D space does not always produce the desired results since evenly spaced streamlines in 3D space do not guarantee visual clarity after they are projected to the screen.

The main idea of our algorithm is that in order to ensure that streamlines will be well organized in the resulting image, it is more effective to place seeds directly on the image plane. Those screen space seed positions can be unprojected back to unique 3D positions in object space by taking into account their depth values. When a streamline is integrated in object space, we make sure that it is not too close to the existing streamlines in image space.

4.1 Evenly Spaced Streamlines in Image Space

To start our algorithm, a random seed is first selected on the image plane and then mapped back to object space. Here, we assume that a depth value is available for every pixel on the screen. Details about different ways to generate the depth map are discussed in the next section. From the initial seed position, a streamline is integrated and placed into a queue Q. We require that all streamlines keep a distance of d_{sep} away from each other on the image plane. To ensure this, the following steps are repeated until Q is empty:

1. Dequeue the oldest streamline in Q as the current streamline.
2. Select all possible seed points on the image plane at a distance d_{sep} away from the projection of the current streamline. Considering each projected sample point on the current streamline, there are two candidate positions for the seeds—one on each side of the streamline.
3. For each of the candidate seeds, a new streamline is integrated as long as possible before it is within the distance d_{sep} from other streamlines on the screen. Then, this new streamline is enqueued.
4. Go back to step 1.

The algorithm above is very similar to the algorithm presented in [7], which works well for 2D flow fields. However, for 3D vector fields, because a projection process from object space to image space is involved, some issues need to be addressed.

4.1.1 Perspective Projection

The algorithm in [7] approximates the distance between a seed point to the nearby streamlines using the distances from the seed point to the sample points of the streamlines, which are the points computed at every step of streamline integration. Those distances will be used to compare with the desired distance threshold d_{sep} to make sure that the streamlines are not too close to each other. The assumption

to make this approximation acceptable is that the distance between the sample points along a streamline must be smaller than d_{sep}. In our algorithm, the streamline distance threshold d_{sep} is defined in image space. Since the integration step size is controlled in object space, after being projected to the screen through perspective projection, the distance between the sample points along a streamline might be shortened or lengthened, which may violate the minimum d_{sep} requirement. To address this issue, for each integration step, the projected distance between two consecutive points on the streamline is computed in image space. If the distance is larger than d_{sep}, some intermediate sample points on the streamline are generated by interpolation.

4.1.2 Depth Comparison

The 3D streamlines can overlap or intersect with each other after being projected to image space. In the 2D evenly spaced streamline algorithm [7], a new sample point on the streamline is invalid when it is within d_{sep} from existing streamlines or when it leaves the 2D domain defining the flow field. In those cases, the streamline will be terminated. In our algorithm, simply terminating a streamline if it is too close to existing streamlines' projections on the image plane is not always desirable because a streamline closer to the viewpoint should not be terminated by those far behind. To deal with this issue, in our algortihm, when the newly generated point of the current streamline is too close to an existing streamline, we first check whether this point is behind that existing streamline. If it is, the integration is terminated. Otherwise, we check whether the streamline segment connected to this new point intersects with the existing streamline on the image plane. If they intersect and the new segment is closer to the viewpoint, the intersected segment of the old streamline becomes invalid and will be removed, and the integration of the current streamline continues. If they do not intersect, the integration of the current streamline continues. In this way, we can ensure that a correct depth relationship between the streamlines is displayed.

4.2 Streamline Placement Strategies

Having described how to control the spacing between streamlines, in this section, we discuss several strategies to place streamlines on the image plane. Since we need to unproject the seeds back to 3D object space to start streamline integrations, depth values for the screen pixels, that is, a depth map, will be needed. In our algorithm, this depth map is generated as a result of rendering 3D objects derived from the input data set, which defines the regions of interest that the user desires. In the following, we present several examples on how the user can explore the flow field.

4.2.1 Implicit Stream Surfaces

Visualizing implicit stream surfaces can be an effective way to explore flow fields since it is known that streamlines are always adhered to the surface and the local flow direction is perpendicular to the normal of the surface. By visualizing different stream surfaces, the user can get a better understanding of the flow field's global structure. Showing only the stream surfaces, however, is not sufficient since no

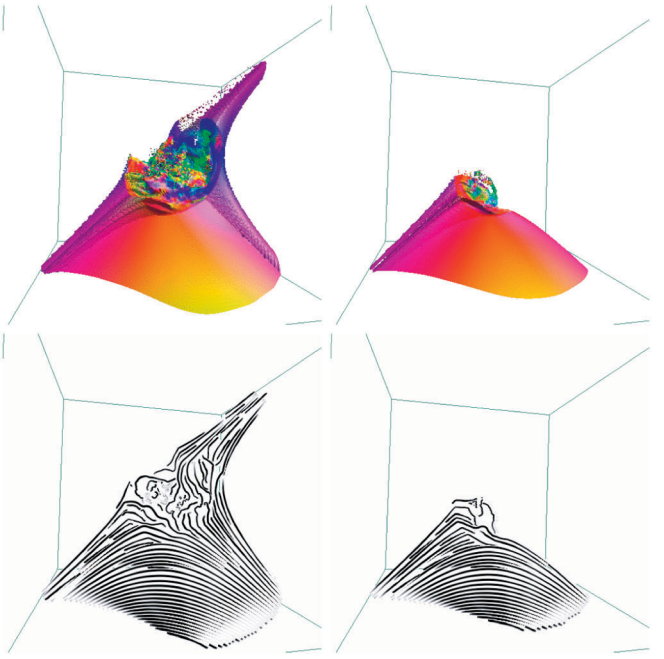


Fig. 3. Streamlines generated on two different stream surfaces.

information about the flow directions on the surface is displayed, as shown in the top images of Fig. 3. To create a more effective visualization, we can first render a stream surface and then use the depth map from the rendered result as the input to our algorithm to create better organized streamlines.

To generate stream surfaces, a volumetric stream function needs to be computed. Previously, van Wijk [22] proposed a method to generate implicit stream functions by computing a backward streamline from every grid point in the volume and recording its intersection point at the domain boundary. If some scalar values are assigned to the boundary, these values from the boundary can be assigned back to the 3D grid points according to the intersection points of their backward streamlines to produce a 3D stream function. Isosurfaces can then be generated from this 3D function to represent the stream surfaces. He proposed "painting" certain patterns on the boundary and seeing how the patterns evolve as the flow goes from the boundary into the domain.

We devise a new method to assign scalar values to the boundary based on preselected streamlines. Our goal is to more clearly visualize the flows in the regions spanned by those streamlines. We first calculate the intersection of those streamlines to the boundary. Then, we treat each intersection point on the boundary as a source of a potential function that emits energy to its surrounding area on the boundary. The energy distribution is set to be a Gaussian function where the intensity is inversely proportional to the distance to the source. For every grid point on the boundary, we then sum up the energy contribution from all sources and use the resulting scalar field on the boundaries to create the 3D implicit stream function. With such setup, we are able to generate stream surfaces enclosing the input streamlines in layers using different

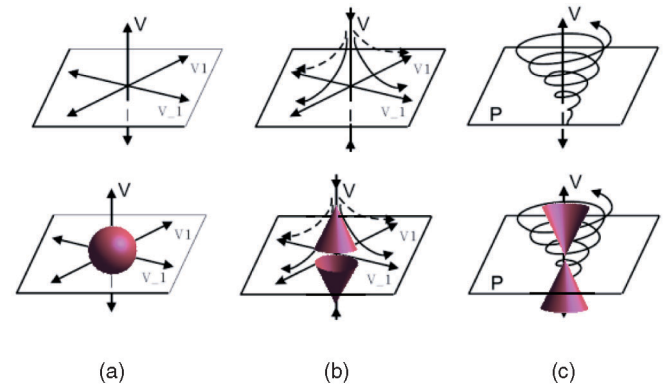


Fig. 4. Seeding templates for different types of critical points. (a) Repelling or attracting node. (b) Attracting or repelling saddle and spiral saddle. (c) Attracting or repelling spiral (critical point classification image courtesy of Alex Pang).

isovalues, and our image space method will place streamlines on each of the stream surfaces to depict the flow directions. Fig. 3 shows two examples of the stream surfaces with different isovalues and the streamlines generated using our method. The data set was generated as part of a simulation that models the core collapse of supernova.

4.2.2 Flow Topology-Based Templates

A great deal of insight about a flow field can often be obtained by visualizing the topology of the field, which is defined by the critical points and the tangent curves or surfaces connecting them. With the topology information, the behavior of flows and, to some extent, the structure of the entire field can be then inferred. Different types of critical points characterize different flow patterns in their neighborhood. Given a critical point, we can compute the eigenvalues and eigenvectors of its Jacobian matrix. The eigenvalues can be used to classify the type of the critical point, whereas the eigenvectors can be used to find its invariant manifold. Previously, Globus et al. [3] proposed to use 3D glyphs to visualize the flow patterns around critical points. Ye et al. [26] proposed a template-based seeding strategy for visualizing 3D flow fields. In short, the method in [26] first identifies and classifies critical points in the 3D field. Then, seeds are placed on the predefined templates around the critical points. Finally, Poisson seeding is used to populate the empty region. The main goal of this method is to reveal the flow patterns in the vicinity of the critical points. To incorporate the idea into our algorithm and to highlight the flow topology, what we need is a depth map that signifies the critical points. We choose to use solid objects to be our templates. Rendering the templates can generate the input depth map for seed placement.

Based on the eigenvalues and eigenvectors, we have different templates for different types of critical points. Fig. 4 illustrates the templates for the following types of critical points. Note that we do not drop seeds directly on the solid object template in object space; we drop seeds on the image by rendering those templates at a given view. So, we only need to decide the shape, orientation, size, and type of the templates, rather than how many seeds to drop or where to drop the seeds:

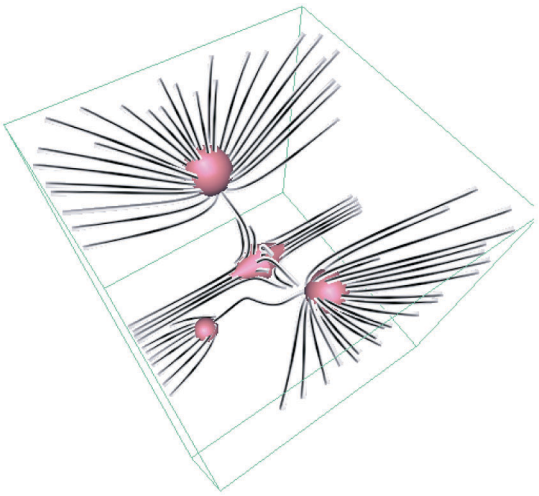


Fig. 5. Streamlines generated from critical point templates. The three sphere templates stand for sinks, whereas the two-cone template stands for saddle.

- *Nodes*. Critical points of this type are sources or sinks of streamlines. The template we use is a solid sphere that is centered at the position of the critical point. The radius of the sphere is scaled by the eigenvalue's real part.
- *Node Saddles*. Two cones are used as the template for this type, which point to the opposite direction of the local eigenplane spanned by the eigenvectors. The radius and height are scaled by the eigenvalue's real part.
- *Spiral*. Two cones are used as the template for this type, which point to each other from the opposite direction of the local eigenplane spanned by the eigenvectors. The radius and height are scaled by the eigenvalue's real part.
- *Spiral saddles*. The template for this type is the same as that of *Node Saddles*.

For a 3D flow field, it is possible that there is more than one critical point. When multiple critical points are present, each critical point has its corresponding template and they are rendered together to the same image. The resulting depth map will have separate regions representing different templates from which seeds are dropped. Fig. 5 shows streamlines integrated from the depth map generated by rendering solid object templates. There are four critical points in this synthetic flow field: three sinks and one saddle.

4.2.3 Isosurfaces of Flow-Related Scalar Quantities

Many scalar variables are related to the properties of a flow field. For instance, vorticity magnitude can often reveal the degree of local rotations, whereas Laplacian can show the second-order derivatives of the flows. As described in [20], those scalar quantities are often important in understanding flow fields even though they are not necessarily directly related to the flow directions. When exploring a flow field, one can first generate images from the isosurfaces of those variables. As the users find some interesting features from the isosurface, they can use our image space method to drop seeds on the screen directly. This allows one to enrich the image and highlight the correlations between the scalar variable and the flow directions. Fig. 6 shows an example of streamlines

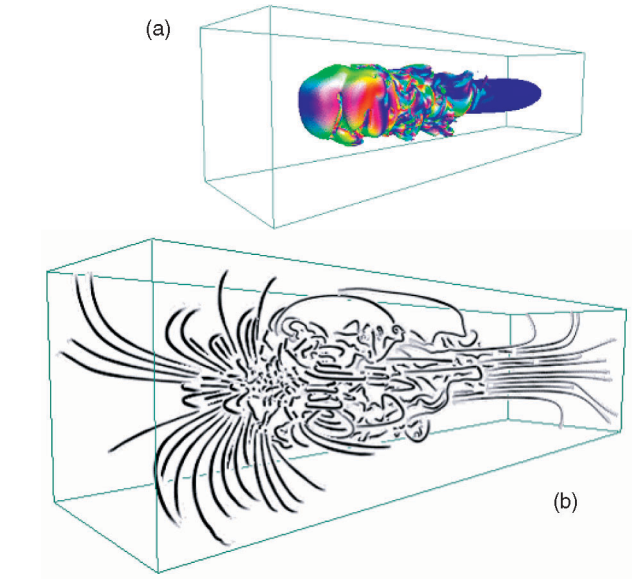


Fig. 6. (a) An isosurface of velocity magnitude colored by using the velocity (u,v,w) as (r,g,b). (b) Streamlines generated from the isosurface with our image space method.

generated from an isosurface of velocity magnitude using our algorithm. The data set was from a simulated flow field of thermal downflow plumes on the surface layer of solar.

4.2.4 Slicing Planes

One effective way to visualize volume data, particularly for regions that are easily occluded, is to slice through the volume and only visualize data on the slice plane. Although slicing planes have been used frequently for visualizing 3D scalar fields, they are not used as often for vector field visualization. One of the primary reasons is that visualizing the vectors only on the plane does not reveal enough insight about the global flow directions, whereas visualizing a large number of streamlines starting from a slice plane can easily clutter the scene. With our image space method, we can enhance the visual clarity by first rendering selected slice planes to the screen, colored with optional scalar or vector attributes, and then dropping seeds on the planes to compute the streamlines. With our spacing control mechanism, we are able to control the depth complexity and show only the outer layer of the streamlines that originated from the slice. Fig. 7 shows an example of streamlines computed from seeds dropped on a slicing plane. Note that the streamlines are computed in 3D space rather than constrained on the slice.

4.2.5 External Objects

Another application of our image space method is in dropping seeds on the surface of a user-selected object. This external object can be thought of as a 3D rake [4], from which streamline seeds are emitted. Although, previously, people have proposed using 3D widgets as seed placement tools, the seeds were explicitly placed on the 3D surface of the widget. This requires an explicit discretization of the rake surface to determine the 3D seed positions. In our image space method, we only need to have a 2D rendered image and depth map of the object. The seed density is determined in image space and thus can be easily adapted to the resolution of images. Fig. 8 shows streamlines

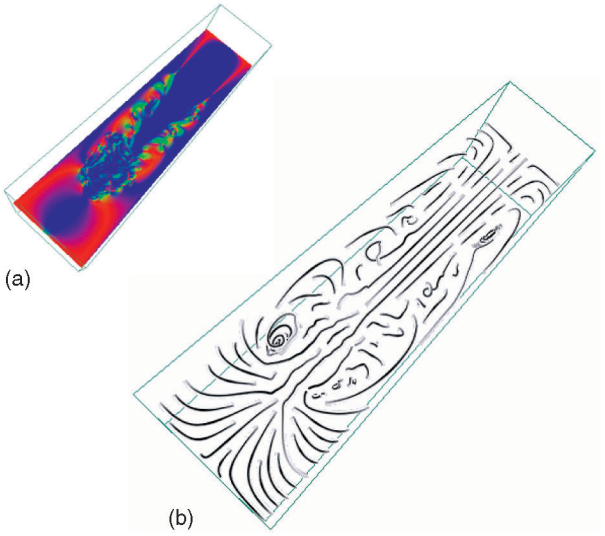


Fig. 7. (a) A slicing plane colored by using the velocity (u,v,w) as (r,g,b). (b) Streamlines generated from the slicing plane with our image space method.

computed from seeds on the surface of a cylinder. Note that, in this image, we enhance the depth cue by mapping the computed streamlines with a texture that enhances the outlines. Details about the rendering are described in Section 4.3.6.

4.3 Additional Runtime Control

In this section, we describe several additional controls and effects that can be achieved using our algorithm.

4.3.1 Level of Detal Rendering

In computer graphics, LOD is commonly used to save unnecessary rendering time for objects whose details are too small to be seen on the screen. Rendering low-resolution

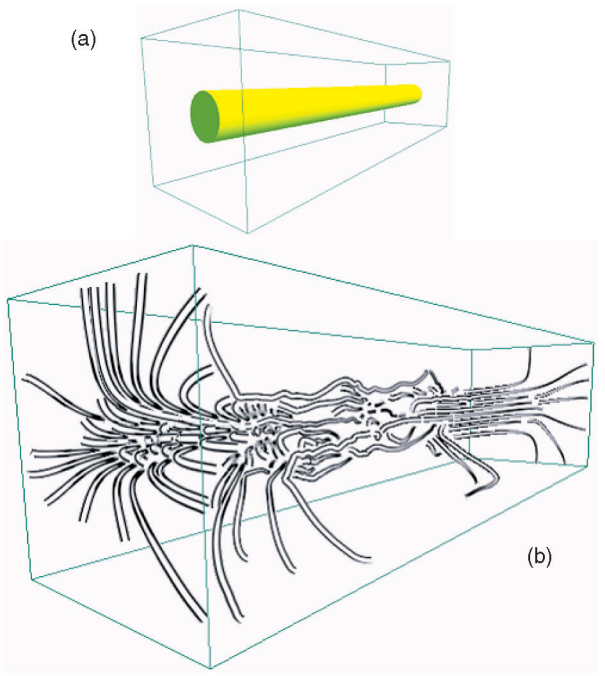


Fig. 8. (a) The cylinder as an external object. (b) Streamlines generated from a cylinder.

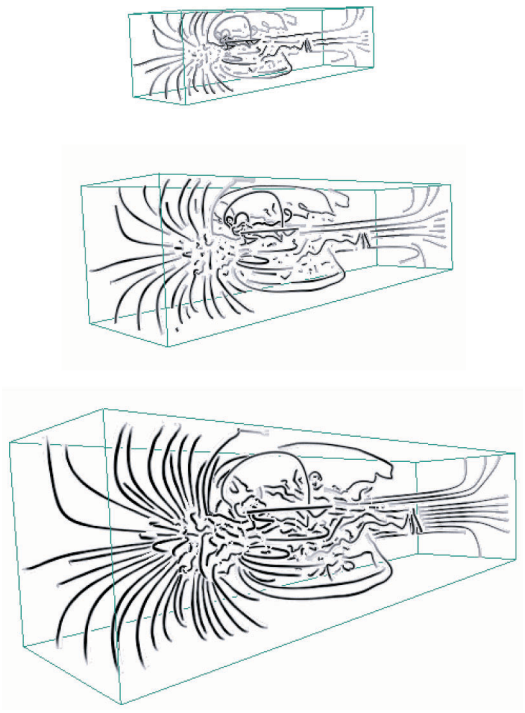


Fig. 9. LOD streamlines generated at three different scales. It can be seen that as the field is projected to a larger area, more streamlines that can better reveal the flow features are generated.

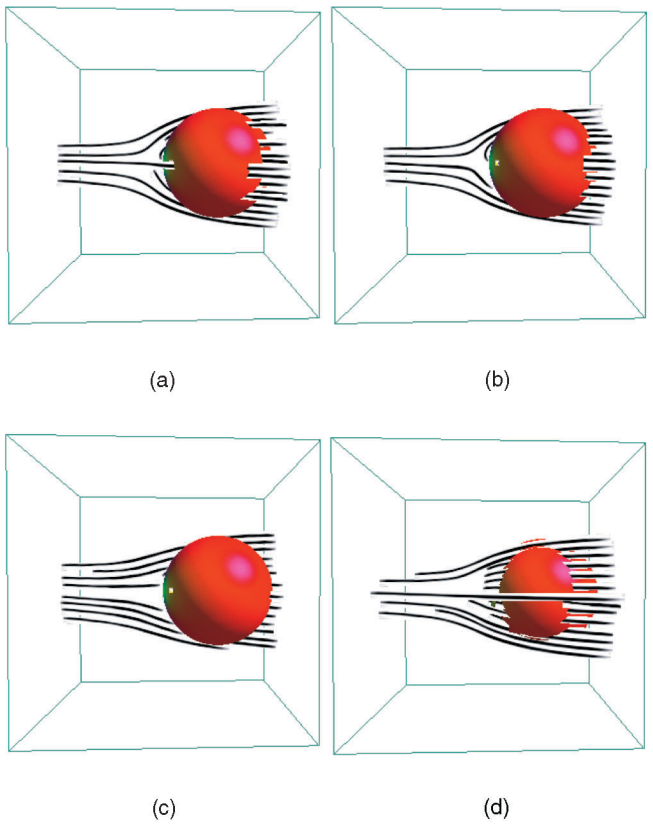


Fig. 10. Streamlines computed using different offsets from a depth map generated by a sphere: (a) from the original depth map, (b) by increasing a value from the original depth map, (c) by further increasing a value from the original depth map, and (d) by decreasing a value from the original depth map.

data can also reduce rendering artifacts if the screen resolution is not high enough to sample the high-frequency detail. One such example is the texture mip-mapping algorithm supported by OpenGL. When visualizing streamlines, to improve the clarity of visualization, Jobard and Lefer [8] proposed to compute a sequence of streamlines with different densities, whereas Mebarki et al. [16] proposed elongating all previously generated streamlines before placing new streamlines when the density is increased. We can adopt the idea of LOD by adjusting the number of streamlines displayed on the screen according to the projection size of the domain on the screen. To achieve this effect, a constant streamline spacing defined in screen space between streamlines is used. As the user zooms out of the scene, because the screen projection area of the domain becomes smaller, fewer streamlines will be generated as a result of attempting to keep the constant distance between streamlines. On the other hand, as the user zooms into the scene, since a larger projection area of the domain is now displayed, more streamlines will be generated and displayed. Fig. 9 shows an example of LOD streamlines generated at different zoom scales.

4.3.2 Temporal Coherence

When the user zooms in and out of, or rotates, the scene, the projection of the surface will be changed. If we rerun our algorithm to generate a completely new set of streamlines whenever such changes occur, some unwanted flickering and other annoyances may happen. To avoid this, we need to maintain temporal coherence for the streamlines generated between consecutive frames. When the user zooms into the surface, the projection area becomes larger. The streamlines from the previous projection need to be retained and placed into the queue as the initial set of streamlines (see Section 4.1). These streamlines are first elongated before new ones are generated. Some sample points along the streamlines may go out of the view frustum and thus become invalid. When the user zooms out, we first verify the sample points along the streamlines from the previous frame and invalidate those points that are too close to other streamlines under the new projection. After this, new streamlines will be added to fill the holes, if any. For rotations, it involves the elongation, validation, and insertion of new lines similar to the zoom operations.

4.3.3 Layered Display of Streamlines

To improve the clarity of visualization, sometimes it is necessary to reduce the rendered streamlines to a few depth layers. One technique related to controlling the depth of rendered scenes is depth peeling [1] for polygonal models. However, depth peeling for lines is not well defined since lines themselves cannot form effective occluders because the spaces between lines are not occupied. Our image space method lends itself well to effective depth control and peeling. This is because seeds are placed on top of the depth map on the image plane. We can "peel" into the 3D flows by slowly increasing or decreasing a δz from the original depth map to drop the initial seeds and generate streamlines. Fig. 10 shows examples where we compute streamlines using different offsets from a depth map generated by a sphere. We can also control the display of streamlines by constraining them to integrate within a $+/-\delta z$ away from the input depth map. This will effectively control the depth complexity of the rendered scene. This is essentially to create clipping planes to remove streamlines outside the allowed depth range. In our case, the clipping planes have shapes

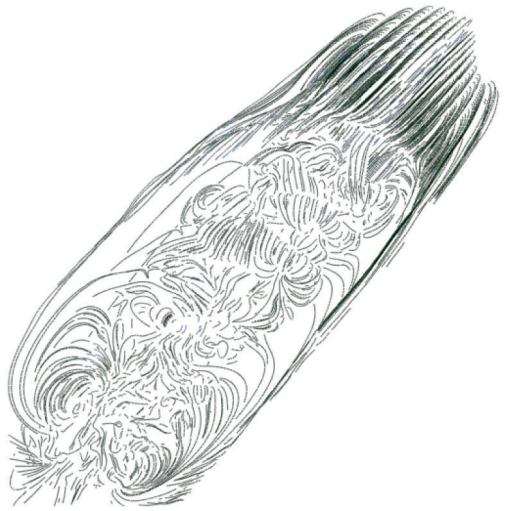


Fig. 11. An example of peeling away one layer of streamlines by not allowing them to integrate beyond a fixed distance from the input depth map.

conforming to the initial depth map, which can be more flexible compared to the traditional planar clipping planes. Fig. 11 shows an example of opening up a portion of the streamlines in the middle section by not allowing streamlines to go beyond a small δz from the input depth map.

4.3.4 Generate Streamlines from Multiple Views

Even though we have focused on how to control the placement of streamlines in image space from a single camera view so far, sometimes, it can be beneficial to combine the streamlines generated from multiple views and display them all together. For each individual view, we still enforce our spacing constraints, that is, not to allow streamlines to come too close to each other. However, when combining the streamlines from multiple views, there is no constraint enforced. The motivation behind this is that, even though the projection of streamlines from different views may intersect and overlap with each other in image space, as long as the depth complexity in each view and the number of combined views are well controlled, the combined streamlines can enhance the 3D depth perception of the scene. In our algorithm, the selection of different views is done by the user: Given a 3D object displayed on the screen, a stream surface, for example, the user can rotate the surface, identify a good view, and place streamlines based on the current view using our image space algorithm. Then, the user can rotate the scene again to reveal the region that was invisible in the previous view and place more streamlines. When the user feels that the scene is getting too cluttered, the accumulation of streamlines can be stopped. Fig. 12 illustrates this process by showing images from four different views and the combined results. Another strategy

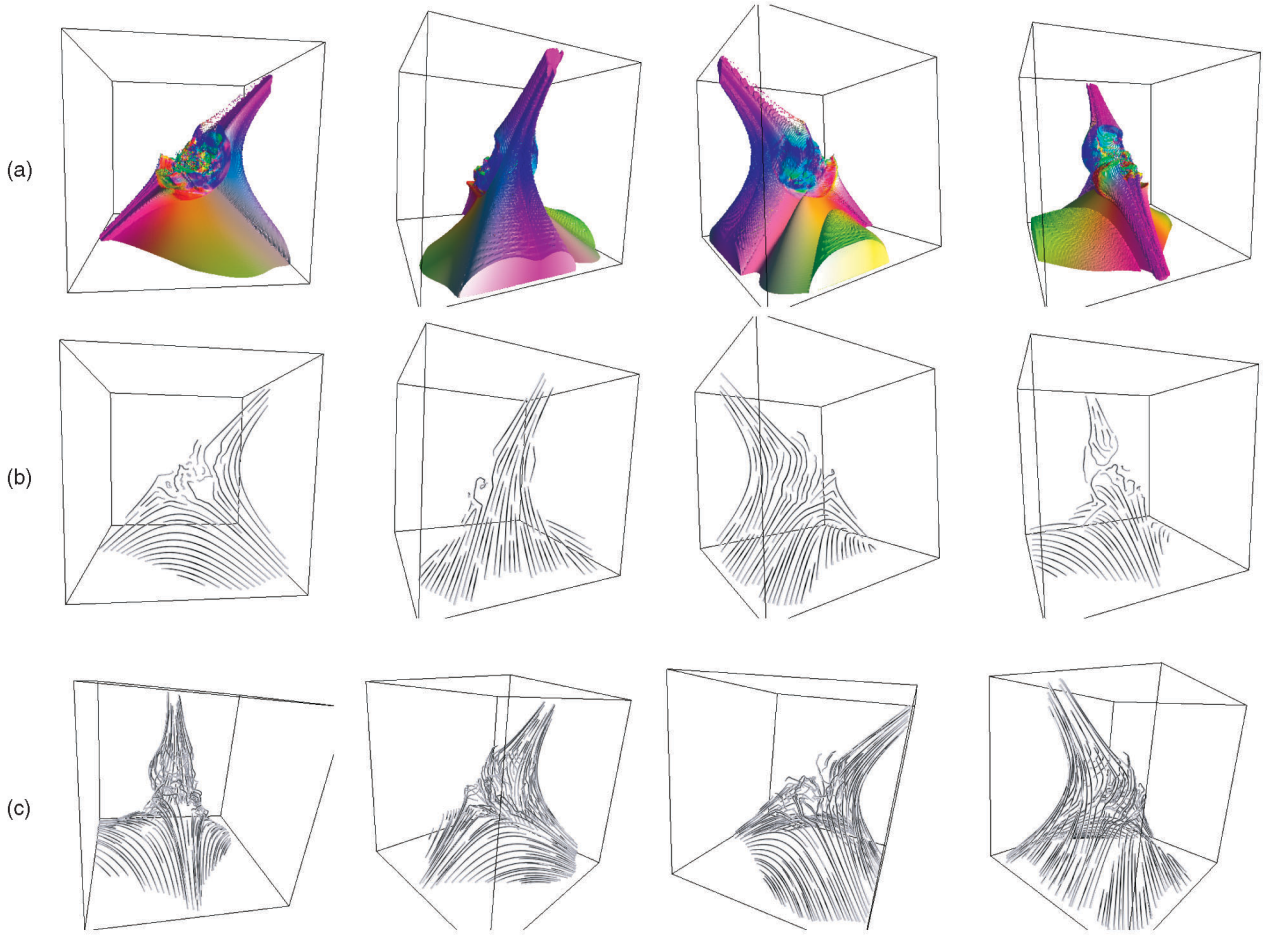


Fig. 12. (a) Rendering images of stream surface from different viewpoints. (b) Streamlines generated at the corresponding viewpoint. (c) Combined images of streamlines rendered from four different views.

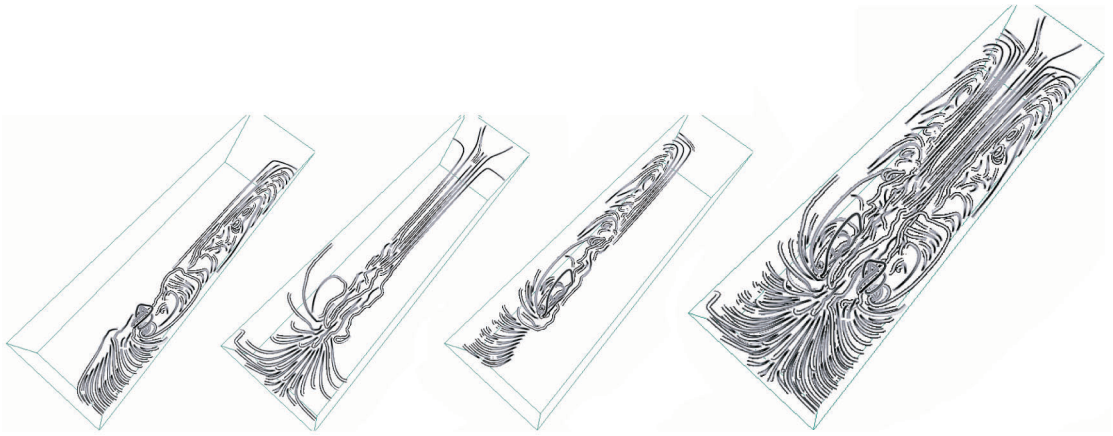


Fig. 13. Streamlines generated from three different cylinder locations (left three images) are combined together and rendered to the image on the right.

for combining the streamlines is to keep the current camera view, but to move the probing objects to different locations. For example, the user can use a cylinder to probe the flow field and place streamlines from the projection of the cylinder surface. The user can keep the current camera view, change the location of the cylinder, and gradually populate the scene until the image reveals enough about the flow field without making the scene overly crowded. Fig. 13 shows an example of combining the streamlines generated from three different cylinder probe locations.

4.3.5 Importance-Driven Streamline Placement

To distinguish regions of different importance, different spacing thresholds can be used to place the streamlines. For instance, more streamlines can be placed in regions with higher velocity magnitudes, whereas fewer streamlines are placed in other regions. To achieve this effect, our algorithm takes an importance map as an input, which can be generated by evaluating any function of the flow field such as velocity or vorticity magnitude. At every step of the streamline integration, we project the point to the screen and then retrieve the importance value at the screen point. After mapping the importance value to a streamline distance, we can decide whether to continue or terminate the integration of the current streamline. We can use different transfer functions to map the importance value to different streamline distance thresholds. For instance, in our implementation, we use the functions of *bias* and *gain* proposed in [18] by Perlin and Hoffert to create nonlinear mapping effects. Fig. 14 shows an example of using the velocity magnitude as the importance value to determine the streamline density on a slice plane, where more streamlines are placed in regions with higher velocity magnitudes.

4.3.6 Stylish Drawing

One advantage of our image-based streamline placement algorithm is that streamlines are well spaced out with user control on the screen. With the spacing controlled, it becomes much easier to draw patches of desired widths along the streamlines on the screen to enhance the visualization of streamlines, since we can easily avoid the stream patches overlapping with each other. To compute the stream patches, we use the screen projection of the streamline as the skeleton. Then, we extend the width of the stream patches along the direction that is perpendicular to the streamline's local tangent direction on the screen. The width of the stream patches is controlled by the local spacing of the streamlines, which is defined by our image-based algorithm. With the stream patches, we can map a variety of textures to enhance depth cues and simulate different rendering styles. We can also vary the width and transparency of the stream patches based on local flow properties. Fig. 15 shows three examples of our stylish drawing of streamlines using different textures.

5 Data Sets and Performance

We have tested our algorithm on a PC with an Intel Pentium M 2-GHz processor, 2 Gbytes of memory, and an nVIDIA 6800 graphics card with 256 Mbytes of video memory. Two synthetic data sets (Figs. 5 and 10) and two 3D flow simulation data sets (Plume and TSI) were used to test our algorithm and generate the images shown throughout the paper. The Plume data set (Figs. 6, 7, 8, 9, 11, 13, 14, and 15) is a 3D turbulent flow field with dimensions of $126 \times 126 \times 512$. The original data set is a time-varying 3D flow field, which models turbulence in a solar simulation performed by National Center for Atmospheric Research scientists. The

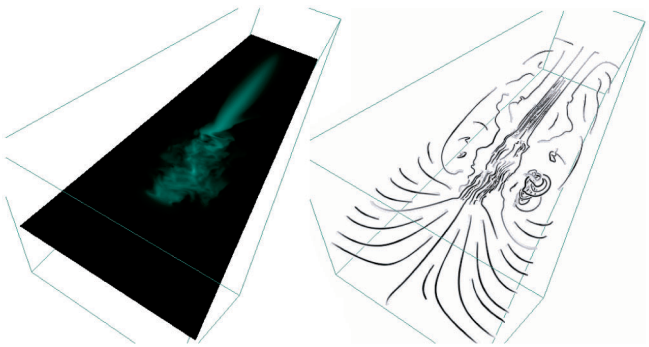


Fig. 14. Streamline densities are controlled by velocity magnitude on a slice. (a) Larger velocity magnitudes are displayed in brighter colors. (b) Streamlines generated from the slice.

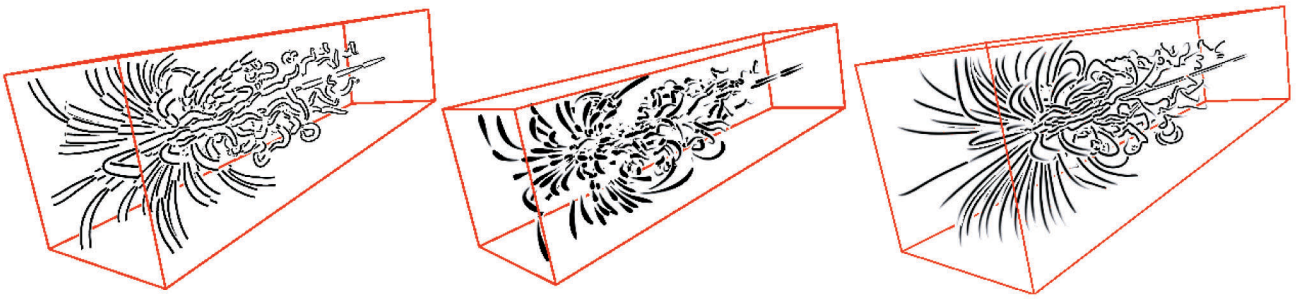


Fig. 15. Streamlines generated and rendered with three different styles by our image-based algorithm.

TSI data set (Figs. 3 and 12) is a 3D flow field with dimensions of $200 \times 200 \times 200$. It was used to model the core collapse of supernova and generated by collaboration among Oak Ridge National Lab and eight universities. We tested our algorithm using a few time steps of these two data sets.

When running our algorithm, the user can control the streamline density by specifying different separating distances in screen space. The coverage of the visualized objects in the input depth map affects the generation of streamlines. The larger the area, the more streamlines are generated compared with those generated from a smaller area if the separating distance remains the same. In our tests, we zoomed into the scene to allow the geometries producing the depth map to cover the screen as much as possible. We used a constant step size Runge-Kutta fourth-order integrator to compute the streamlines.

We show the performance of our algorithm using the Plume data set. The main steps of our algorithm include transformations of streamline points between object and image space, streamline integration, seed point selection, and validation of streamline points. In our program, since longer streamlines were generally preferred, we discarded those streamlines that were too short. In our experiments, the threshold value for the minimum streamline length was 20, and the fixed integration step size was 1.0, both in voxels. This means that, for a streamline to be accepted, it should have at least 20 integration points. The larger this threshold value, the higher the possibility for a streamline generated from our distance control algorithm to be discarded becomes; thus, the percentage of total computation time wasted on generating those short streamlines becomes higher, too. Fig. 16 shows the percentages of time spent on each of the main steps in our algorithm. In the figure, it can be seen that the streamline integration process is the most time-consuming part. Fig. 17 shows the number of streamlines and line segments generated with different distance thresholds. Fig. 18 shows the timings for generating streamlines with different separating distances, which directly influence the number of streamlines that were computed. Throughout the paper, all images of streamlines are rendered with stylish drawings; the average time to render one line segment is about 0.00259 millisecond.

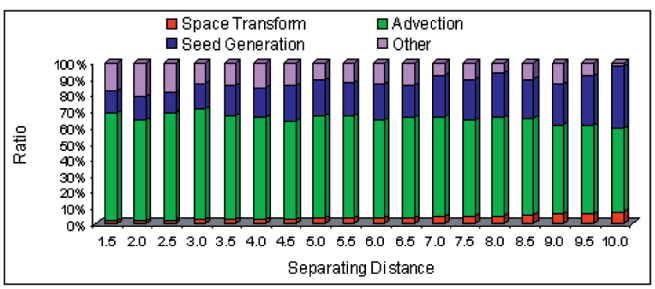


Fig. 16. The percentage of total time each main step used.

6 Conclusions and Future Work

We present an image-based approach for streamline generation and rendering. Our main goal is to reduce scene cluttering and allow the user to flexibly place streamline seeds on the screen when they identify hot spots from the visualization of other scalar or flow-related variables. The rendering output from a variety of visualization techniques such as isosurfaces or slicing planes can be used as the input to our program to assist seed selections. As streamlines are integrated in object space, our algorithm monitors and controls their distances

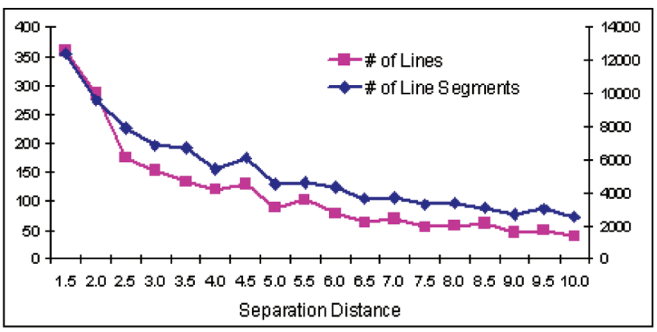


Fig. 17. The pink curve (the left axis as scale) shows the number of streamlines, whereas the blue one (the right axis as scale) shows the number of line segments generated.

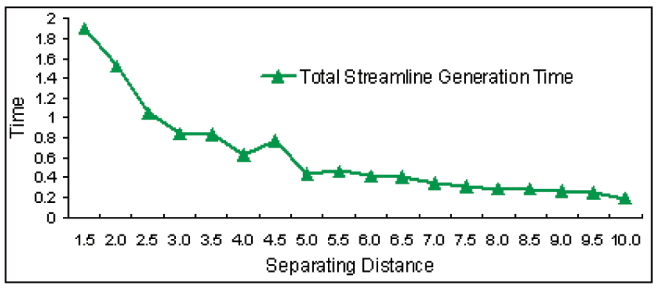


Fig. 18. The time (in seconds) to generate streamlines from an isosurface for different separating distance (pixels) using the Plume data set.

to the existing streamlines that have already been displayed. In addition to reducing visual cluttering, our algorithm can be used to achieve a variety of effects such as LOD rendering, depth layering, and stylish drawing of streamlines.

In the future, we will focus on designing different image-based strategies to detect domain-specific flow features. We will also improve the rendering of streamlines generated by our algorithm with additional depth cues. Finally, we will apply our algorithm to generate various nonphotorealistic rendering effects to have a better illustration of 3D vector fields.

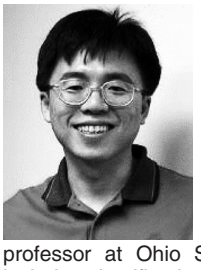
ACKNOWLEDGMENTS

This work was supported by US National Science Foundation ITR Grant ACI-0325934, US NSF RI Grant CNS-0403342, DOE Early Career Principal Investigator Award DE-FG02-03ER25572, and US NSF CAREER Award CCF-0346883. The Plume data set is courtesy of the National Center for Atmospheric Research, and the TSI data set is courtesy of Oak Ridge National Lab. The authors would like to thank the reviewers for their helpful comments.

REFERENCES

[1] C. Everitt, *Interactive Order-Independent Transparency.* 2001.
[2] L. da Vinci, *Old Man Seated on Rocky Outcrop, Seen in Profile to the Right, with Water Studies,* Windsor Castle, Royal Liberty, RL. 12579r, c. 1510–1513.
[3] A. Globus, C. Levit, and T. Lasinski, "A Tool for Visualizing the Topology of Three-Dimensional Vector Fields," *Proc. IEEE Conf. Visualization (VIS '91),* pp. 33-40, 1991.
[4] K.P. Herndon and T. Meyer, "3D Widgets for Exploratory Scientific Visualization," *Proc. Seventh Ann. ACM Symp. User Interface Software and Technology (UIST '94),* pp. 69-70, 1994.
[5] A. Hertzmann and D. Zorin, "Illustrating Smooth Surfaces," *Proc. 27th Ann. Conf. Computer Graphics and Interactive Techniques (SIGGRAPH '00),* pp. 517-526, 2000.
[6] V. Interrante and C. Grosch, "Strategies for Effectively Visualizing 3D Flow with Volume LIC," *Proc. IEEE Conf. Visualization,* pp. 421-424, 1997.
[7] B. Jobard and W. Lefer, "Creating Evenly Spaced Streamlines of Arbitrary Density," *Proc. Eurographics Workshop in Boulogne-sur-Mer (Visualization in Scientific Computing '97),* 1997.
[8] B. Jobard and W. Lefer, "Multiresolution Flow Visualization," *Proc. Ninth Int'l Conf. Computer Graphics, Visualization and Computer Vision (WSCG '01),* 2001.
[9] R. Laramee et al., "Texture Advection on Stream Surfaces: A Novel Hybrid Visualization Applied to CFD Simulation Results," *Proc. Eurographics/IEEE-VGTC Symp. Visualization '06 (EuroVis '06),* 2006.
[10] R. Laramee, B. Jobard, and H. Hauser, "Image Space-Based Visualization of Unsteady Flow on Surfaces," *Proc. IEEE Conf. Visualization '03,* pp. 131-138, 2003.
[11] G.-S. Li, U. D. Bordoloi, and H.-W. Shen, "Chameleon: An Interactive Texture-Based Rendering Framework for Visualizing Three-Dimensional Vector Fields," *Proc. IEEE Conf. Visualization (VIS '03),* 2003.
[12] O. Mallo et al., "Illuminated Lines Revisited," *Proc. IEEE Conf. Visualization,* pp. 19-26, 2005.
[13] X. Mao et al., "Image-Guided Streamline Placement on Curvilinear Grid Surfaces," *Proc. IEEE Conf. Visualization (VIS '98),* pp. 135-142, 1998.
[14] O. Mattausch et al., "Strategies for Interactive Exploration of 3D Flow Using Evenly Spaced Illuminated Streamlines," *Proc. 19th Spring Conf. Computer Graphics (SCCG '03),* 2003.
[15] N. Max, R. Crawfis, and C. Grant, "Visualizing 3D Velocity Fields Near Contour Surfaces," *Proc. IEEE Visualization '94,* pp. 248-255, 1994.
[16] A. Mebarki, P. Alliez, and O. Devillers, "Farthest Point Seeding for Efficient Placement of Streamlines," *Proc. IEEE Conf. Visualization,* pp. 479-486, 2005.
[17] S. Park et al., "Dense Geometric Flow Visualization," *Proc. Eurographics/IEEE-VGTC Symp. Visualization (Data Visualization '05),* 2005.
[18] K. Perlin and E.M. Hoffert, "Hypertexture," *Proc. 16th Ann. Conf. Computer Graphics and Interactive Techniques (SIGGRAPH '89),* pp. 253-262, 1989.
[19] D. Stalling and M. Zockler, "Fast Display of Illuminated Field Lines," *IEEE Trans. Visualization and Computer Graphics,* vol. 3, no. 2, 1997.
[20] N.A. Svakhine et al., "Illustration and Photography Inspired Visualization of Flows and Volumes," *IEEE Conf. Visualization,* 2005.
[21] G. Turk and D. Banks, "Image-Guided Streamline Placement," *Proc. 23rd Ann. Conf. Computer Graphics and Interactive Techniques (SIGGRAPH '96),* pp. 453-460, 1996.
[22] J.J. van Wijk, "Implicit Stream Surfaces," *Proc. IEEE Conf. Visualization '93 (VIS '93),* pp. 245-252, 1993.
[23] J.J. van Wijk, "Image-Based Flow Visualization," *Proc. 29th Ann. Conf. Computer Graphics and Interactive Techniques (SIGGRAPH '02),* pp. 745-754, 2002.
[24] J.J. van Wijk, "Image-Based Flow Visualization for Curved Surfaces," *Proc. IEEE Conf. Visualization '03 (VIS '03),* 2003.
[25] V. Verma, D.T. Kao, and A. Pang, "A Flow-Guided Streamline Seeding Strategy," *Proc. IEEE Conf. Visualization,* pp. 163-170, 2000.
[26] X. Ye, D.T. Kao, and A. Pang, "Strategy for Seeding Three-Dimensional Streamlines," *Proc. IEEE Conf. Visualization,* pp. 471-478, 2005.

Liya Li received the BE and MS degrees from the Department of Computer Science and Engineering, Beijing Institute of Technology, China, in 1999 and 2002, respectively. Currently, she is a PhD candidate in the Department of Computer Science and Engineering, Ohio State University. Her primary research interests include visualization and computer graphics.



Han-Wei Shen received the BS degree from the Department of Computer Science and Information Engineering, National Taiwan University, in 1988, the MS degree in computer science from the State University of New York at Stony Brook in 1992, and the PhD degree in computer science from the University of Utah in 1998. From 1996 to 1999, he was a research scientist at the NASA Ames Research Center in Mountain View, California. He is currently an associate professor at Ohio State University. His primary research interests include scientific visualization and computer graphics. He received the US Department of Energy's Early Career Principal Investigator Award and the US National Science Foundation's CAREER Award. He also received the Outstanding Teaching Award from the Department of Computer Science and Engineering, Ohio State University.