Image Based Streamline Generation and Rendering

Liya Li*
The Ohio State University

Han-Wei Shen†
The Ohio State University

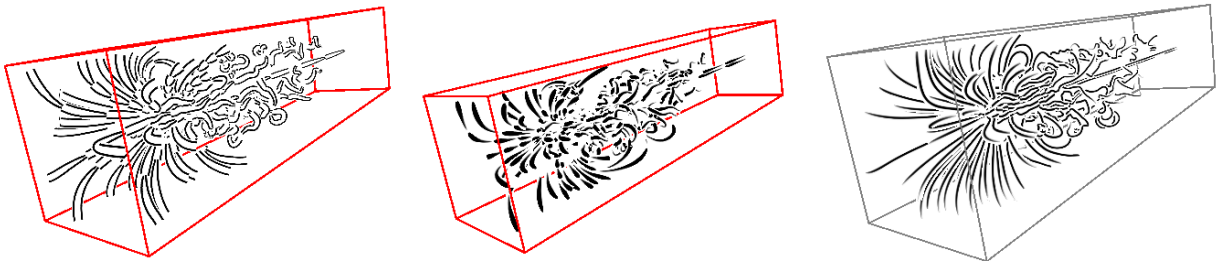


Figure 1: Streamlines generated and rendered with three different styles by our image based algorithm

ABSTRACT

Seeding streamlines in 3D flow fields without considering their projections in the screen space can produce visually cluttered rendering results. Streamlines will overlap or intersect with each other in the final image, which makes it difficult for the user to perceive the underlying flow structure. This paper presents a method to control the generation and rendering of streamlines in the image space to avoid visual clutter and allow a more flexible exploration of the flow fields. In our algorithm, two dimensional images 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 the streamlines can be flexibly controlled based on various criteria to further improve visual clarity. With the image space approach, it becomes easy to implement level of detail rendering and depth peeling of streamlines to allow for more effective visualization of 3D flow fields.

Keywords: streamlines, Image based, Image space, Object space, 3D flow visualization

1 INTRODUCTION

Effective visualization of 3D vector fields plays an important role in many scientific applications since many simulations now produce both scalar and vector data. It is often necessary to have a comprehensive understanding of those data in order to obtain insight into the underlying problem. To visualize three dimensional vector fields, researchers in the past have developed a variety of techniques. Those techniques include geometry based methods such as streamlines and particle animations, which resemble what the scientists are used to see in their experiments, 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, Spot Noise, or the more recent Image Based Flow Visualization (IBFV) [16] techniques are mostly suitable for visualizing two dimensional flows. When extending those techniques to three dimensions, occlusion becomes a major problem and they require additional means to identify regions of interest before texture can be mapped. There is also a challenge to extend the texture based methods to unstructured grids although the recent work by Van Wijk [17] and Laramee et al. [7] have provided excellent solutions to map flow textures to arbitrary surfaces.

Compared to texture based approaches, visualizing streamlines or pathlines is still more frequently used in various scientific applications because it is more flexible to place the lines around regions of interest and render them at interactive speeds. The challenge for visualizing streamlines/pathlines, however, is that the scene can quickly get cluttered when too many of them are displayed. In addition, visualizing three dimensional points and lines has more perceptual difficulties compared to visualizing surfaces. It is much harder to add three dimensional depth cues to points and lines since they do not form effective occluders to reduce clutter in the scene. Also it is much more difficult to show a coherent structure from the lines even when lighting [13] is applied. Previously, researchers have attempted to develop effective seed placement algorithms for the purpose of better visualizing the streamlines as well as enhancing salient flow structures [5, 14, 12, 18]. However, most of the algorithms were developed for two dimensional vector fields which cannot be directly applied to visualizing three dimensional data.

In this paper, we present an image based approach for streamline generation and rendering. To better display three dimensional streamlines and reduce visual cluttering, there is a need to control how the streamlines will distribute across the image after they are projected to the screen. To achieve this goal, instead of placing streamline seeds directly in the three dimensional space, we carefully select seeds from the image plane and then unproject the seeds back to the object space before streamline advection takes place. The three dimensional positions of the unprojected seeds can be uniquely identified by taking as an input a two dimensional image and its corresponding depth map. Those two dimwnsional images are generated by the user on the fly as she/he is exploring flow related scalar quantities. By carefully spacing out streamlines in the image plane as they advect, we can effectively reduce visual cluttering and minimize the depth ambiguity caused by overlapping streamlines in the final image. With the two dimensional image, the seeds can also be more effectively and flexibly placed. As the user is exploring data using additional visualization techniques, when she/he spots interesting features on the screen, the seeds can be directly dropped on the image without the need to have a separate process to search for three dimensional regions of interest corresponding to those features. Our image space streamline placement

*e-mail: lil@cse.ohio-state.edu
†e-mail:hwshen@cse.ohio-state.edu

algorithm lends itself well to achieve a variety of effects such as level of detail rendering and depth peeling. We can also render the streamlines in a variety of styles to enhance the perception of the three dimensional flow lines.

The remainder of the paper is organized as follows. We first briefly review the existing work on seed placement and streamline rendering. We then introduce our image based seed placement and streamline generation algorithm. We present a variety of ways to utilize our image based algorithm for better visualization of three dimensional streamlines. We then discuss strategies to generate the input depth images to assist the process of streamline generation.

2 Related Work

Streamlines rendering is still one of the most popular techniques for visualizing flow fields. For 2D flow fields, there are several techniques available for generating seeds to reduce visual cluttering. The image guided streamline placement algorithm proposed in [14] uses an energy function to measure the difference between a low-pass filtered version of the image and the desired visual density. Then an iterative process is used to reduce the energy through some pre-defined operations on the set of streamlines. For creating evenly spaced streamlines, Jobard et al. [5] proposed to control the distance between two adjacent streamlines to achieve the desired density. Both the seed selection and the termination of advection are controlled by first measuring the distance to the existing streamlines and then determining the desired actions. Mebarki et al. [12] proposed a 2D streamlines seeding algorithm by placing a new streamline at the farthest point away from all existing streamlines. Verma et al. [18] applied various templates around critical points to capture important flow patterns. Then a Poisson disk distribution is used to randomly distribute additional seed points in those non-critical regions. This work has been extended to 3D flow field in [19].

Generating desirable and aesthetically pleasing streamlines in three dimensional flow fields is much more difficult than in two dimensional fields, because the projection process from the object space to the image space can cause overlapping and intersection, which is impossible for two dimensional streamlines. Ye et al. [19]presented a strategy for streamline seeding through analyzing the flow topology. Critical points are used to identify the flow regions with important pattern, 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 *et al.*'s algorithm [5] was extended to 3D in [10]. Spatial perception of the 3D flow was improved by using depth cueing, halos. And also they applied focus+context methods, ROI driven streamline placing, and spotlights to solve the occlusion problem.

There have been some techniques proposed for the rendering of 3D flow fields for a better perception of the spatial information. Lighting is one of the elements to improve spatial perception. Stalling et al. [13] employed a realistic shading model to interactively render a large number of properly illuminated field lines by using 2D texture and texture transformation. The algorithm is based on a maximum lighting principle, which gives a good approximation of specular reflection. To improve diffuse reflection, Mallo et al. [9] 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 continuities. Interrante et al. [3] used a visibility-impeding 3D volumetric "halo function"to define the locations and strengths of the gaps making the depth discontinuities.

3 Algorithm Overview

In this paper, we present a novel method to control scene cluttering when visualizing 3D streamlines by placing streamline seeds in the image plane in such a way that the depths and structures of streamlines can be clearly presented. Figure 2 shows the visualization pipeline of our image-based streamline generation scheme. The input to our algorithm includes a 2D image and its corresponding depth map, and a 3D vector field that the user is interested in exploring. The 2D input images can come from a variety of sources. They can be generated from 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 different scalar quantities. Our algorithm will generate streamlines by placing seeds on the image plane following certain criteria mentioned in the following sections of the paper. With the depth values of the selected pixels, those seeds can be unprojected back to the 3D space. Streamlines are then advected in the 3D object space. Our algorithm ensures that streamlines will not come too close to each other after they are projected to the screen.

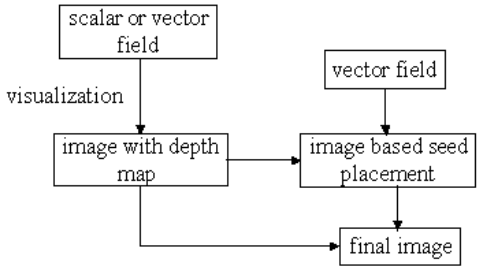


Figure 2: Visualization pipeline of our image based streamline generation scheme.

There are several major advantages for placing seeds on the image plane. One is to avoid scene cluttering caused by streamlines having a very high depth complexity. Placing streamlines in 3D positions without considering their screen projections often causes visual cluttering, as 3D streamlines may intersect with or come arbitrarily close to each other in the image plane. Although researchers have previously proposed to draw haloed lines to resolve ambiguity of streamline depths [10], when a large number of short line segments generated from the haloing effect are displayed, the relative depth relationship between the streamlines is still very difficult to comprehend. By appropriately controlling the spacing of seeds as well as the streamlines themselves in the image plane, we are able to prevent the visualization from being overly crowded. As will be seen in the later sections, placing seeds in the image space also allows us to render the streamlines in layers, which will significantly reduce the depth complexity of the streamlines displayed and hence enhance the understanding of 3D flow structures.

Another advantage of the image based approach is that it enhances the understanding of correlation between the underlying flow field and other simulation variables. When analyzing a flow field, the user often needs to visualize other variables as well in order to obtain a comprehensive view of the underlying physical problem. Dropping seeds at the regions of interest defined by certain properties can assist creating a better mental model to comprehend the data. Traditionally, visualization of streamlines and other scalar properties are often performed independently. Dropping streamline seeds in region that is defined by other variables requires an explicit search of the regions of interest in 3D space. In our work, instead of considering the region of interest specification and seed placements as two separate processes, we intend to provide a unified way by allowing the user to drop seeds whenever they identify interesting features in the image generated during a visualization session. This makes the seed placement process much more flexible since there is no need for the algorithm to know what is being rendered in the im-

age. As a result, the process of issuing queries to answer the user's hypotheses both in scalar and vector fields can be done in a more coherent manner.

To implement our idea, one key issue to resolve in our research is how we are going to place seeds and advect streamlines from 2D images so that the complexity of visualization is carefully controlled. We are also interested in exploring a variety of ways to generate and utilize the input 2D images and depth maps to assist us creating a better visualization of streamlines. In the following, we discuss those issues in detail.

4 Image Space Streamlines Placement

The primary goal of our research is to generate well organized streamlines in the image space from a 3D vector field. A key research question to ask is how we can control the streamline spacing in the 2D image when we are rendering 3D streamlines. Previously, researchers have proposed several techniques to place streamlines for 2D vector fields that can ensure an even or near even spacing [14, 5, 12]. Researchers have also proposed to extend those ideas to 3D fields by ensuring an evenly spaced streamlines in 3D [10]. Unfortunately, a straightforward extension of 2D streamline placement methods for 3D vector fields does not work well since, for example, evenly spaced streamlines in 3D space does not guarantee evenly spaced streamlines in their 2D projection.

A key observation of our research is that in order to ensure that the streamlines be well organized or evenly spaced out in the resulting image, it is better to place the seeds directly on the image plane. The control of streamline spacing should also be performed in the image space. Since any screen pixel corresponds to an infinite number of 3D points, the 3D screen points that define the seeds need to be unprojected back to some 3D points. This is achieved by using an input depth map resulted from other 3D visualization techniques. We defer the discussion of possible ways to generate the depth map in later sections. In this section, we only focus on the placements of seeds and streamlines. With the idea of choosing seeds and controlling streamlines in the image plane, it becomes apparent to us that we are able to leverage the previous work in 2D streamline spacing algorithms with a moderate amount of modification. At present, our algorithm adopts the idea presented in the evenly-spaced streamlines algorithm proposed by Jobard el at. [5], but we are not limited to this algorithm only. For example, the technique proposed in [12] can most likely work equally well or better. In the following, we discuss our algorithm.

4.1 Evenly-Spaced Streamlines in 3D

In order to have a good control of the density of streamlines in the image space, seed selection and termination of streamline advection are very important. In our algorithm, these two steps are performed in the image space, while the advection is done in the object space. This means we do not project the 3D vector field into 2D image space and perform advections.

In our algorithm, at the beginning, a random seed is selected in the projection region of a 3D surface in image space. This seed is mapped back to object space after interpolating a depth value from the depth map. The initial streamline is integrated from this seed and placed into a queue Q. We will defer our definition of the validness of sample points in section 4.1.2. We assume that all streamlines need to be d_{sep} apart from each other in the image plane. 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 at d_{sep} apart from the projection of the current streamline. For each projected sample point, there are only two possible candidate points, one at each side of the streamline
3. Integrate new streamlines from the valid candidate seeds as long as possible before they are within d_{sep} from other streamlines in the image space

The algorithm above is very similar to the 2D algorithm in [5], which works well for 2D flow fields. However, for 3D vector fields, because there is a projection process from the object space to the image space involved, some problems need to be taken care of.

4.1.1 Perspective Projection

The algorithm in [5] approximates the distance between a seed point to the nearby streamlines using the distance from the seed point to the sample points of the streamlines. The assumption to make this approximation acceptable is that the distance between sample points along a streamline must be smaller than d_{sep}. In our algorithm, as the integration step size is controlled in object space, after being projected to screen through perspective projection, the distance between sample points along streamlines might be shortened or elongated, which may violate the minimum d_sep requirement. We proposed two ways to solve this problem.

The first method is achieved by adding more check points between two sample points. For each integration step, the projected distance between two consecutive points in advection is computed in image space. If the distance is larger than d_{sep}, some intermediate sample points are generated by interpolation, and the corresponding image space points are inserted into the grid based structure if they are still d_{sep} apart from other streamlines. This checking process takes place after integrating the current point, and before accepting the new sample points.

The first method uses a fixed step size to perform integration. In the case that an adaptive step size is more desired, for each step of the integration, we first project the velocity at the current point to then image space, and get an approximate step size by advecting some distance smaller than d_{sep} in the image space. We then use this step size to integrate the point.

4.1.2 Depth Comparison

Streamlines can overlap or intersect with each other after being projected to image space, unless the depth map is generated from a stream surface. Although the property of stream surface guarantees that streamlines on the surface never intersect with each other, our image based algorithm is not limited to use only stream surfaces as the source to generate the 2D input depth map. In the 2D algorithm [5] a new sample point is not valid if 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 as a result. When extending this algorithm to 3D, only checking the distance between the sample points with existing streamlines in the image space is not sufficient. This is because a streamline closer to the the viewpoint should not be terminated by the one far behind, even when the newly generated sample point is too close to the existing streamline. To deal with this issue, in our algorithm, when the newly generated sample point is too close to an existing streamline, we check whether the line segment from this newly sample point intersects with the existing streamline. If they intersect and the new segment is closer to the viewpoint, the segment of the existing streamline becomes invalid, and the advection of the current streamline continues. If they do not intersect, the advection continues. If the newly sample point is far behind the existing streamline, the advection is terminated. In this way, we can ensure a correct depth relationship between streamlines in the projection space. Figure 3 shows an example of our evenly spaced streamlines on a flow field generated from a data set simulating the core collapse of supernovas.

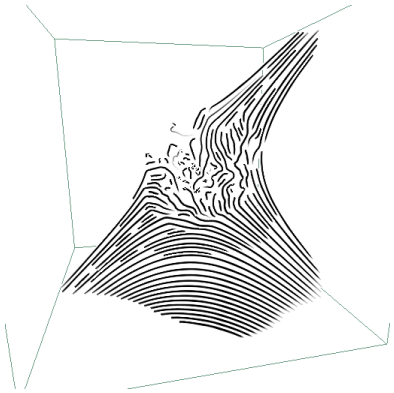


Figure 3: Evenly-spaced streamlines on a stream surface.

4.2 Importance Driven Seed placement

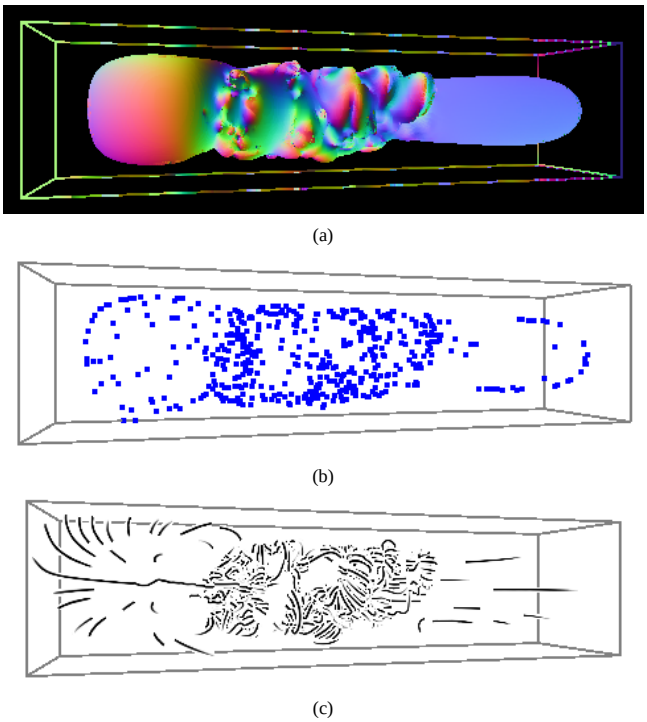
Although the algorithm described above works well for maintaining a set of evenly-spaced streamlines in the image plane, it is not always suitable when there is a need to drop more seeds in certain hot spots in the image plane. This is because in the algorithm above, which was primarily derived from the 2D algorithm proposed by Jobard *et al.*[5], the placement of seeds only considers the requirement of maintaining particular spacing, which is less flexible if other factors such as region importance need to be considered. In fact, seed placement and the control of streamline spacing can be considered separately. Streamlines can be computed from seeds generated in the image plane based on any other criteria, and spacing control of streamlines can be done after the streamlines are computed. In this section, we present an alternative method to achieve this goal.

To generate seeds based on region importance, we assume that the pixels in the input 2D image will have intensity values proportional to their importance. For instance, if the goal is to visualize flows in regions which have higher local velocity gradients, we can color the geometry to be rendered based on this property and then render the geometry to the screen. Then, seeds can be generated in the image plane where more seeds are placed in regions of higher importance and fewer seeds are placed elsewhere. We achieve this through a random seed placement process that reflects the region importance as follows. To place each seed, we first select a random position (x,y) within the image plane. Then we generate another random number v within the range of image intensity and then compare this number with the intensity i at (x,y) from the importance image. If the random number v is larger than i, we discard this seed otherwise we place the seed in (x,y). Assuming there is an equal probability for the random number v to be anywhere in the image intensity range, this process will stochastically generate more seeds in regions which are more important, i.e., with a higher image intensity.

After all seeds are generated using the stochastic process, streamlines are generated although not drawn to the screen. To adequately control the spacing, during the process of generating the streamlines, we divide the screen into a regular lattice and then deposit all sample points of streamlines to the lattice element. After the generation of streamlines is completed, we choose a limited amount of streamlines to display in the screen with a goal not to violate the minimum spacing requirement. This is done by walking through every points of the streamline to check the local neighborhood in the lattice whether there are already streamline points being drawn previously. Streamline segments that are within the minimally allowed radius will not be displayed. To favor longer streamlines, we traverse the streamlines from the longest one in a descending order since streamlines drawn earlier will prevent from lines that come later from growing. We also take into account the 3D depth comparison issue as described in the previous section.

The fundamental difference between the algorithm here from the previous one is that we decouple the process of seed generation and streamline spacing control to offer additional flexibility. Instead of carefully planning where to drop the next seed so as not to violate the spacing constrain, we generate streamlines from any given seed sets and delete the streamlines that violate the spacing requirement afterwards. Here our focus is more on avoiding scene cluttering rather than guaranteeing all streamlines are evenly spaced out in an absolute sense since most applications do not have such a strong constrain that all streamlines should be evenly spaced out. Our algorithm can also allow us to perform a more global control as opposed to being purely greedy in the sense that if the goal is to generate longer streamlines, we can more easily put effort to retain those longer ones since we have the whole set of streamlines to make the selection.

Figure 4(c) shows an example of our importance driven scheme applied on a simulated flow field of thermal downflow plumes in the surface layer of Solar. Figure 4(a) displays the vectors using a false color map on a velocity magnitude isosurface. Seeds are focused on regions with higher velocity gradients as shown in Figure 4(b). The spacing of streamline advections is also controlled by the velocity gradient as in Figure 4(c).



(a)

(b)

(c)

Figure 4: Importance driven seed placement: image (a) displays vectors by mapping (u,v,w) to (red, gree, blue) on a velocity magnitude isosurface. (b) more seeds are dropped at the places with higher local velocity gradients evaluated in the image space. (c): Streamline spacing is controlled by the local velocity gradients, where a shorter distance is used for regions that have higher velocity gradients

4.3 Run Time Control of Streamline Generation

In this section, we discuss some additional controls of streamline placements to further enhance the clarity of visualization.

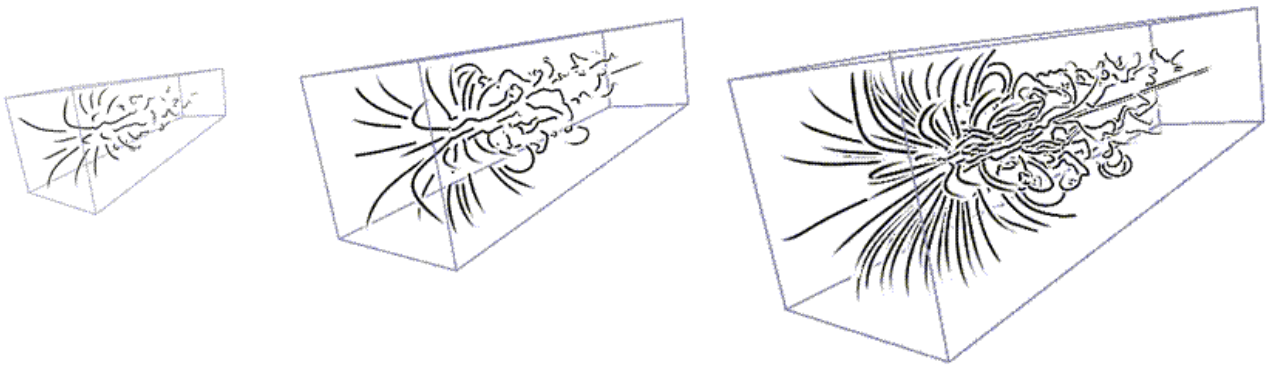


Figure 6: Level of detail streamlines generated at three different scales. It can be seen that as the field is projected to a larger area, more streamlines that can better reveal the flow features are generated.

Figure 5: An example of peeling away one layer of streamlines by not allowing them to advect beyond a fixed distance from the input depth map.

4.3.1 Layered Display of Streamlines

One major problem when visualizing 3D streamlines is that streamlines may overlap with each other in the image plane after projection. A streamline scene with a high depth complexity can cause difficulties in perceiving the underlying 3D structure of the flows. Previously, researchers have proposed to use haloing effect when rendering the streamlines. However, halos can only alleviate the scene cluttering problem to a very limited extent as it is still difficult to make sense of a large number of broken line segments when the underlying depth complexity of the scene is too high. To effectively control the clarity of the visualization, it is necessary to reduce the rendered streamlines to a few depth layers. One type of techniques that is related to controlling of the depth of rendered scene 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 space between lines are not occupied.

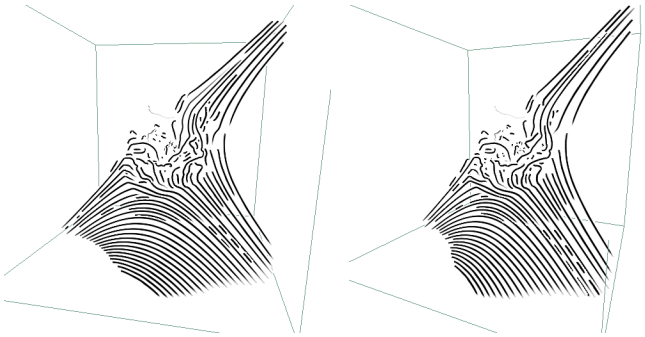
Our image space method lends itself well to effective depth control and peeling. This is because seeds are placed on top the depth map in the image plane. We can "peel" into the 3D flows by slowly increasing a δz from the original depth map to drop the initial seeds and generate the streamlines. We can also control the display of streamlines by constraining them to advect within a +/- δz 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 conforming to the initial depth map. This can be more powerful compared to the traditional planar clipping planes. Figure 5 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.

4.3.2 Level of Detail Rendering

Level of detail is often used as an effective way to control scene complexity as well as avoid unnecessarily computation. Computing and displaying streamlines at subpixel positions in the image plane does not add any value to the visualization other than causing cluttering and aliasing. Our image space method can be used to generate level of detail display of streamlines automatically. When a fixed image plane spacing between streamlines are specified, as we zoom in and out of the scene, more and fewer streamlines will be generated in the domain automatically since the seeds are placed in the screen space and the space between streamlines will never goes below a prescribed distance. Figure 6 shows an example of the streamlines generated at different zoom scales.

4.3.3 Temporal Coherence

As the user zooms in and out, or rotates the scene, the projection area of the surface will also be changed. A brute force algorithm is to generate new streamlines whenever such changes occur. However, in this way, some unwanted flickering and other annoyances might happen. To avoid these, temporal coherence of the rendered streamlines must be ensured. When the user zooms into the surface, the projection area becomes larger. The streamlines from 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 might be out of the view frustum and thus becomes invalid. When the user zooms out, we validate the sample points along the streamlines from previous projection and mark those invalid points. After this, new streamlines will be added in, even though some of them may be short streamlines. For the rotation operation, it involves both the elongation and validation and insertion of new lines similar to zooming in and out. Figure 7 shows examples of streamlines generated during a continuous rotation.



(a) (b)

Figure 7: Our image based method preserves temporal coherence. Images (a) and (b) were generated when the user rotated the scene. Notice similar lines are present in those two images.

4.3.4 Stylish Drawing

One advantage of our image based streamline placement algorithm is that the streamlines are well spaced out in the image plane. With the spacing controlled, it becomes much easier to draw patches of a desired width 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 streamlines as the skeletons. 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 known to our image based algorithm. With the stream patches, we can map a variety of textures to simulate different rendering styles. We can also vary the width and transparency of the stream patches based on local flow properties. Figure 1 shows three examples of our stylish drawing of streamlines.

4.4 Strategies for Generating Depth Maps

In this section, we discuss several strategies for generating the depth maps, which can be used as the input to our streamline generation algorithm to guide the user visualizing the flow fields.

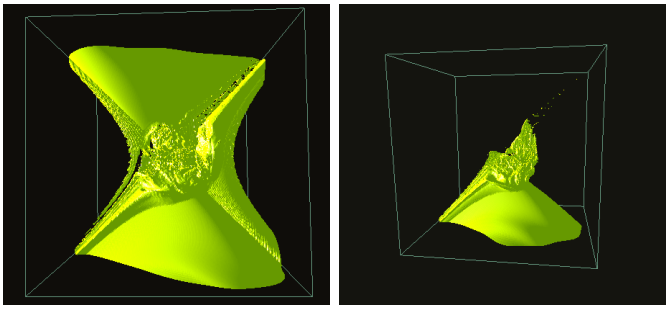
4.4.1 implicit stream surfaces

Stream surfaces are the surfaces whose normals are everywhere perpendicular to the local flow directions. In other words, every point on the surface satisfies the following rule:

$$\vec{N} \cdot \vec{V} = 0 \tag{1}$$

where N is the normal of the stream surface at the point and V is the vector. Previously, Van Wijk [15] proposed a method to generate implicit stream surfaces by computing a backward streamline from every grid point in the volume and recording its intersection point at the domain boundary. If a scalar field is assigned to the boundaries, values from the boundaries can be assigned to the grid points according to the intersection points of their backward streamlines. Isosurfaces can then be generated from this scalar field to represent the stream surfaces.

Images of rendering stream surfaces can be used as an input to our image based algorithm. It has a unique property that streamlines lie entirely on the surface and will not cross each other even in the image space. By rendering implicit stream surfaces in layers using different isovalues, we can place seeds on the surfaces to visualize streamlines in layers.



(a) **(b)**

Figure 8: Images of stream surfaces generated by using different isovalue.

In the case that the user computes a large number of streamlines in 3D space in a separate process, we can use implicit stream surfaces and our image based method to create a better visualization of those streamlines. This can be done by first generating the implicit stream function derived from the streamlines, and then visualizing the stream surfaces in layers and dropping more seeds on the image plane to enhance the understanding of the streamline structures. To create the implicit stream function, instead of assigning scalar values to the boundaries without considering the streamlines to be visualized, we first calculate the intersection of those streamlines to the boundaries. Then we treat each intersection point as a source point of a potential function that emits energy to its surrounding boundary points. The energy distribution can be calculated using a Gaussian function.

Each intersection point or energy source receives a weight based on their locations. We assign a higher weight for points that are toward the center of the intersection point group and lower weights to the outer ones, again, using a gaussian function.

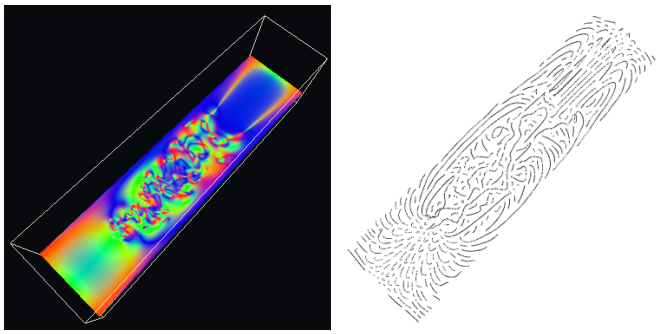
Then we calculate the scalar value for every boundary point by summing up the energy contribution from all sources and use the resulting scalar field on the boundaries to create the 3D implicit stream function. With such setup, we are able to create stream surfaces enclosing the streamlines in layers and drop more seeds on each of the layer to enhance the streamline structure.

Figure 8 show two examples of the stream surfaces using our method.

4.4.2 Slicing planes

Slicing planes have been used as a common way to visualize 3D scalar fields as an alternative (perhaps more popular) way compared to direct volume rendering or isosurfacing. Although simple, in some cases it is more effective since it does not suffer from occlusion or depth ambiguity. Slicing is used less often in visualizing 3D vector fields, however, because projecting the vectors onto the slices may not show enough information about the 3D flow structure while compute streamlines originated from the slicing plane may still generate cluttered scenes. With our image space method, we can first generate an arbitrary slicing plane and map appropriate property to the slice. After render the slice to the image plane, we can select screen space points on the slice based on the scalar values on the slices and then emit streamlines from those points. Using our streamline generation algorithm, we are able to control the depth complexity and show only the outer layer of the streamlines without intersection. We can control the seed placement on

the image plane to control the layering. Figure 9 shows an example of streamlines computed from seeds dropped on a slicing plane.



(a) **(b)**

Figure 9: (a) A slicing plane colored by the velocity field. (b) Streamlines advected from the slicing plane using our algorithm.

4.4.3 External Objects

Another application of our image space method is to drop seeds on the surface of an external object. This external object can be thought of as a 3D rake [2] from which seeds of streamlines are emitted. While previously people have proposed to use 3D widgets as seed placement tool, the seeds were explicitly placed to the 3D surface of the widget. This require an explicit discretization of the rake surface to determine the 3D seed positions. In our image space method, we only need to have the rendered image of the object/widget but not its geometric representation. ¿From the image and the depth map, the user can place seeds on the image space and streamlines are computed from the unprojected seed points in 3D. This way, any image/depth map can be used to drop seeds. Figure 10(a) shows streamlines computed from seeds originated from the surface of a cylinder on the image plane.

4.4.4 Flow Field Related Scalar Quantities

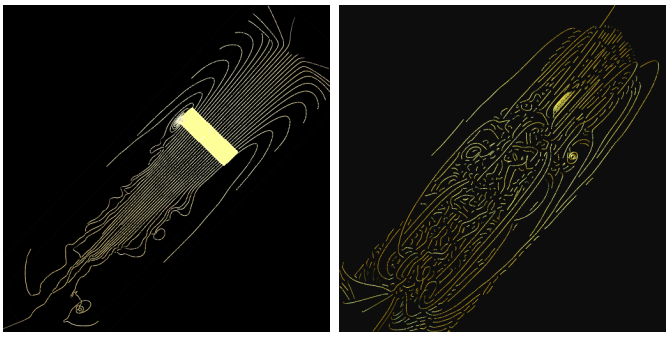
Many scalar quantities are related to the properties of flow fields. For instance, vorticity magnitude can often reveal the degree of local rotations, while Laplacian can show the second order derivatives of the flows. These scalar quantities are important to reveal the flow structures although they are not related directly to the flow directions. Using our method, we can first generate images from the scalar techniques such as isosurface of those variables. To enrich the image and highlight the correlations between those properties to the underlying flow directions, we can drop seeds in the image space and visualize the results together. Figure 10(b) shows an example of dropping seeds on the surface of a vorticity magnitude isosurface on the image plane.

5 Implementation and Performance

We have tested our algorithm on a PC with an Intel Pentium IV 2 GHz processor, 768 MB memory, and an nVIDIA 6600 GT graphics card with 128 MB of video memory.

5.1 Dataset

Two 3D flow data sets were used to test our program and show the images througout the paper - Plume and TSI. Plume is a 3D turbulent flow field with dimensions of 512x512x512. The original data is a time-varying 3D flow field, which was to compute the turlence in a solar simulation done NCAR scientists. TSI data is a 3D flow



(a) **(b)**

Figure 10: Streamlines generated from seeds dropped on a cylinder at different place in the image plane.

field with dimensions of 200x200x200. It was 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 steps of these two data sets.

5.2 Timing

Since our method is image based streamline generation scheme, we chose to use separating distance in screen space between points to estimate the density of streamlines. A 3D flow field is much more complicated than 2D fields, in particular for turbulence data. A major part of time spent on generating streamlines include validation for streamline points, 3D to 2D projection, and streamline advection. Our seed selection and streamline advection algorithms was implemented on a CPU, and the illumination of streamlines was implemented on GPU. We separate the discussion of timing for these two data sets.

Time	Separating dist.	number of streamlines
0.156	12.0	59
0.547	8.0	122
1.203	6.71	146
2.375	3.48	354
4.578	2.12	923

Table 1: The time and number of streamlines for different separating distance (in seconds) on Plume data set.

Time	Separating dist.	number of streamlines
0.281	12.0	64
0.516	9.2	127
1.25	6.22	281
3.546	3.24	492
4.578	2.12	923
7.218	2.74	1102

Table 2: The time and number of streamlines for different separating distance (in seconds) on TSI data set.

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 drop 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 input to our program to assist seed selections, and unprojecting the seeds back to the three dimensional space. As streamlines are advected in the object space, our algorithm monitors and controls their distances to other existing streamlines that have already been plotted. Besides reducing visual cluttering, our algorithm can be used to achieve a variety of effects such as importance driven seed placement, level of detail rendering, depth layering, and stylish drawing of streamlines. We also allow a variety of ways for the user to createtwo dimensional images as input.

Future work will be on experimenting a wider range of input images for detecting domain specific flow features. We will also enhance the depth cues of the streamlines generated by our algorithms. Finally, we will apply our algorithm to generate various non-photorealistic rendering effects to have a better illustration of three dimensional vector fields.

ACKNOWLEDGEMENTS

The authors would like to acknowledge John Clyne at National Center for Atmospheric Research and Oak Ridge National Lab for providing the PLUME and TSI data.

REFERENCES

[1] C. Everitt. Interactive order-independent transparency. Technical report, NVIDIA Corporation, 2001.

[2] K. Herndon and T. Meyer. 3d widgets for exploratory scientific visualization. In *ACM Symposium on User Interface Software and Technology*, 1994.

[3] Victoria Interrante and Chester Grosch. Strategies for effectively visualizing 3d flow with volume LIC. In *IEEE Visualization*, pages 421–424, 1997.

[4] B. Jobard and W. Lefer. Unsteady flow visualization by animating evenly-spaced streamlines. *Computer Graphics Forum (Proceedings of Eurographics 2000)*, 19(3), 2000.

[5] Bruno Jobard and Wilfrid Lefer. Creating evenly-spaced streamlines of arbitrary density. In W. Lefer and M. Grave, editors, *Visualization in Scientific Computing '97. Proceedings of the Eurographics Workshop in Boulogne-sur-Mer, France*, pages 43–56, Wien, New York, 1997. Springer Verlag.

[6] R. Laramee, C. Garth, and H. Schneider, J.and Hauser. Texture advection on stream surfaces: A novel hybrid visualization applied to cfd simulation results. In *Proceedings of EuroVis 2006*. IEEE Computer Society Press, 2006.

[7] R. Laramee, B. Jobard, and H. Hauser. Image space based visualization of unsteady flow on surfaces. In *Proceedings of Visualization 2003*, pages 131–138. IEEE Computer Society Press, 2003.

[8] G.-S. Li, H.-W. Shen, and U. Bordoloi. Chameleon: An interactive texture-based rendering framework for visualizing three-dimensional vector fields. In *Proceedings of Visualization 2003*. IEEE Computer Society Press, 2003.

[9] O. Mallo, R. Peikert, C. Sigg, and F. Sadlo. Illuminated lines revisited. In *IEEE Visualization*, pages 19–26, 2005.

[10] Oliver Mattausch, Thomas Theul, Helwig Hauser, and Meister Eduard Gröller. Strategies for interactive exploration of 3d flow using evenly-spaced illuminated streamlines.

[11] N. Max, R. Crawfis, and Ch. Grant. Visualizing 3D velocity fields near contour surfaces. In *IEEE Visualization '94 Proceedings*, pages 248–255. IEEE Computer Society, 1994.

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

[13] D. Stalling and M. Zockler. Fast display of illuminated field lines. *IEEE Transactions on Visualization and Computer Graphics*, 3(2), 1997.

[14] G. Turk and D. Banks. Image-guided streamline placement. In *Proceedings of SIGGRAPH '96*, pages 453–460. ACM SIGGRAPH, 1996.

[15] J. van Wijk. Implicit stream surfaces. In *IEEE Visualization*, pages 245–252, 1993.

[16] J van Wijk. Image based flow visualization. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 745–754. ACM Press, 2002.

[17] J. van Wijk. Image based flow visualization on curved surfaces. In *Proceedings of Visualization 2003*, pages 123–131. IEEE Computer Society Press, 2003.

[18] Vivek Verma, David T. Kao, and Alex Pang. A flow-guided streamline seeding strategy. In *IEEE Visualization*, pages 163–170, 2000.

[19] Xiaohong Ye, David T. Kao, and Alex Pang. Strategy for seeding 3d streamlines. In *IEEE Visualization*, pages 471–478, 2005.