A Volume Rendering Framework for Visualizing 3D Flow Fields

Hsien-Hsi HSIEH**, Liya LI**, Han-Wei SHEN** and Wen-Kai TAI***

** Department of Computer Science and Engineering, The Ohio State University
Columbus, OH 43210, USA

*** Department of Computer Science and Information Engineering, National Dong Hwa University
Hualien, Taiwan

Abstract

In this paper, we present a volume rendering framework for visualizing 3D flow fields. We introduce the concept of coherence field which evaluates the representativeness of a given streamline set for the underlying 3D vector field. Visualization of the coherence field can provide effective visual feedback to the user for incremental insertion of more streamline seeds. Given an initial set of streamlines, a coherence volume is constructed from a distance field to measure the similarity between the existing streamlines and those in their nearby regions based on the difference between the approximate and the actual vector directions. With the visual feedback obtained from rendering the coherence volume, new streamline seeds can be selected by the user or by a heuristic seed selection algorithm to adaptively improve the coherence volume. An improved volume rendering technique that can render user-defined appearance textures is proposed to facilitate macro-visualization of 3D vector fields.

Key words: Flow Visualization, Volume Rendering, Seed Placement, Voxelization

1. Introduction

To analyze three dimensional vector fields generated from numerical simulations, many techniques have been proposed in the past. Geometry-based methods (such as glyph, hedgerhog, streamline, stream surface, flow volume, etc) use shape, color, and animation to indicate vector directions in the proximity of user-specified regions of interest. Texture-based methods such as Line Integral Convolution (LIC), Spot Noise, and Image Based Flow Visualization (IBFV) display the directional information with synthesized textures to reveal the vector field’s global structure. Generally speaking, texture-based methods can be easily exploited by 2D applications to generate effective visualization. For three dimensional data, however, it is much more challenging due to occlusion and loss of information caused by 2D projection.

Visualizing 3D vector fields using streamlines is still widely used because they are easier to compute and faster to render. However, the projected streamlines can easily clutter the image when too many lines are displayed simultaneously. In addition, it is difficult to differentiate and perceive the spatial relationships among streamlines even with color or lighting because of the lack of depth cues. Although many effective seed placement algorithms were developed, most of those algorithms are not suitable for 3D vector fields. Volume rendering techniques were proposed in the recent years to enhance 3D depth perception, but the effectiveness of visualization highly depends on the complexity of the data.

The primary goal of this research is to develop an end-to-end framework to visualize three dimensional vector fields using volume rendering. There are two benefits of using volume rendering for visualizing three dimensional vector fields. One is that volume rendering can generate “macro” visualization to reveal the flow field’s global structures. The other is the depth attenuation resulted from evaluating the volume rendering equation provides better
cues and can more effectively display the spatial relationships among flowlines. The proposed framework parameterizes a solid three-dimensional volume at a pre-set resolution based on an input streamline set derived from the underlying vector field. The parameterized volume is used to look up user-selected appearance textures to render the directional information. Previous methods such as the Chameleon system\(^{(6)}\) relies on user-input streamlines to conduct volume parameterization. However, there is no guarantee whether the user-input streamlines are representative for the vector field. In this paper, we present a novel scheme based on the concept of \textit{coherence field} to evaluate the representativeness of a given streamline set. The coherence field is a scalar field that indicates the local flow coherence around the existing streamlines. The coherence value of a voxel in the coherence volume is low if the flow direction is very different from what we can infer from the nearby streamlines. By displaying the coherence field of a given set of existing streamlines, visual feedback is provided to indicate where in the field more streamlines are needed. In the case that an automatic algorithm is more desired to generate the streamline seeds, in this paper we present a heuristic algorithm that can utilize the information computed in the coherence field to generate the most representative streamlines for the vector field.

To visualize the vector field, the streamlines generated at every iteration of the update process are converted into an intermediate representation called \textit{trace volume}. A trace volume is an intermediate representation for the vector field consisting of three-dimensional texture coordinates for each voxel and a distance to the nearest streamline. Different from the technique presented in\(^{(6)}\), we propose a new method to create the trace volume based on a distance field computation which allows a more complete and reliable voxelization of the streamlines and their proximity. With the coherence field and the trace volume representations, a variety of depth cueing techniques can be applied, and the volume appearance can be controlled by the user at run time.

The rest of paper is organized as follows. Related works are reviewed in section 2. The framework of volume construction and streamline generation algorithm is presented in section 3. In section 3.1, we introduce the concept of coherence field. Volume rendering with appearance control and user interaction are discussed in section 4. The experimental results are illustrated in section 5. Finally, we conclude the proposed approach and point out some of the future work.

2. Related Work

For 3D vector field visualization, various techniques have been proposed in the past. Numerical methods and three dimensional geometry primitives have been used, such as streamlines. Mattausch et al.\(^{(9)}\) extended the evenly-based streamline placement algorithm proposed by Jobard and Lefer\(^{(5)}\) to 3D flow fields, where the distances between streamlines are controlled. Seeds are selected by some pre-set distance away from existing streamlines, and the advection of a streamline is terminated if it comes too close to the existing streamlines. Ye et al.\(^{(20)}\) proposed a flow topology-based algorithm by placing various templates around critical points to reveal important flow features. For those regions uncovered by the influence regions of critical points, Poisson disk distribution is used to randomly distribute additional seed points. Li and Shen\(^{(7)}\) presented a view-dependent image-based approach to generate streamlines from the 3D vector field. They select seeds and trace streamlines in 3D but in the mean time project streamlines to the view plane to avoid cluttering on image.

In addition to the traditional techniques, there exist algorithms that use volume rendering to visualize 3D flow fields. Crawfis and Max\(^{(4)}\) utilized a volume splatting technique to render 3D vector fields. Rezk-Salama et al.\(^{(11)}\) introduced an effective method to visualize 3D LIC data. Interactive control mechanisms, such as transfer functions and clipping planes are used to explore the interior structures. Li et al.\(^{(6)}\) proposed an interactive texture-based technique to visualize 3D steady and unsteady flow fields. They decouple the calculation of streamlines and the mapping of visual attributes into two separate states, which allows the
user to compute three-dimensional flow textures interactively. Telea and van Wijk\textsuperscript{(15)} proposed 3D IBFV by extending 2D texture-based IBFV\textsuperscript{(18)} for visualizing 3D flow fields with moving texture patterns. Helgeland and Andreassen\textsuperscript{(2)} presented a fast method to compute volumetric LIC textures. A limb darkening technique is used to enhance the spatial relationship between fieldlines, which is implemented by using appropriate transfer functions. Helgeland and Elboth\textsuperscript{(3)} introduced an interactive texture-based method to visualize 3D time-varying flow fields. Evenly-distributed particles are traced along the pathlines to depict the global flow features. In the mean time, those particles are used as seeds to generate instantaneous fieldlines to show the local information in each time frame.

Besides the above techniques for visualizing 3D flow fields, there are other works proposed to further enhance spatial cueing. Zockler\textsuperscript{(13)} proposed a realistic shading model based on a maximum lighting principle to illuminate a large number of field lines interactively using 2D textures. Their method gives a good approximation of specular reflection. In order to improve diffuse reflection, Mallo et al.\textsuperscript{(8)} proposed a view-dependent lighting model on thin cylindrical tubes based on averaged Phong/Blinn lighting. A simplified expression of cylinder averaging is used. Interrante and Grosch\textsuperscript{(4)} used a visibility-impeding 3D volumetric halo function to highlight the locations and strengths of depth discontinuities. Stoll et al.\textsuperscript{(14)} presented an algorithm to visualize stylized line primitives represented as generalized cylinders. Shadow effect is implemented in their algorithm to improve the depth perception.

3. An End-to-End Volume Rendering Framework

Our end-to-end framework for volume rendering of vector fields consists of three key stages: streamline generation, coherence volume generation and analysis, and trace volume creation and rendering. The central idea of this framework is to provide the user with direct visual feedback so that streamlines can be incrementally updated and rendered using volume rendering techniques. Figure 1 shows the flow chart of the framework and the interaction between the framework and the user. Given a vector field as input, the user first computes an initial set of streamlines. Then, an analysis is performed to test whether the inserted streamlines are representative for the vector field. This is done through the computation of coherence volume. Given the initial streamlines, we compute approximate vectors for the grid points in the volume based on their nearby streamlines. The coherence volume is a scalar field that indicates the errors between the approximate vectors and the true vectors from the original vector field. The coherence value of a voxel in the coherence volume is low if it is very different from what we can infer from the nearby streamlines. The coherence volume is rendered to the user through interactive volume rendering or slicing. With visual feedback, the user can insert more streamlines or focus on local regions that have higher errors to refine the visualization. We also design a heuristic algorithm to select seeds based on the coherence field to automate the process. To visualize the vector field, the streamlines generated at every iteration of the

![Fig. 1 Framework of vector field visualization.](image-url)
update process are converted into an intermediate representation called trace volume. A trace volume is an intermediate representation of the vector field consisting of three dimensional texture coordinates for each voxel. The texture coordinates for the voxels along the same streamline path are correlated. Different from the technique presented in (6), we devise a new trace volume creation algorithm that provides a continuous representation for the trace volume based on distance field calculation. Once the trace volume is created, user-defined appearance textures can be used to visualize the vector field with enhanced depth cues. In the following, we discuss each stage of our volume rendering framework in detail.

3.1. Coherence Field and Coherence Volume

Given a set of streamlines in a vector field, we can evaluate how representative the streamline set is based on whether the flow in each streamline’s proximity shows a similar behavior. When the direction of flow is the main concern, the flow behavior can be characterized by the streamline path. In other words, if the flow directions are locally coherent with the given streamlines in the nearby regions, the streamlines are said to be representative for the vector field. Otherwise, more streamlines or a new set of streamlines are needed. Based on this idea, given a vector field and a streamline set as input, we propose a vector coherence metric. With the metric, for a given point in the field, we calculate the difference between the true vector at this point from the input vector field, and a vector inferred from the nearby streamline. When evaluating this coherence metric everywhere in the field, a scalar field, called coherence field, is defined. The concept of coherence field can be used to create a discrete volume called coherence volume, which can be used to show where to insert more streamlines in the field. In the following, we first present the concept of coherence measurement.

Coherence Measurement: Coherence field \( C \) is generated from a set of streamlines \( S \) and a vector field \( V \). For a point \( p \) in the field, a scalar value called vector coherence can be derived. The vector coherence measures how similar the vector \( v_p \) at \( p \) is to the vector \( v_s \) defined at the nearest point on the given streamline set \( S \). This can be computed with a dot product of the two normalized vectors:

\[
c(p) = \frac{v_p \cdot v_s}{|v_p||v_s|} \tag{1}
\]

The vector coherence is in range \([-1, 1]\).

Since in practice a streamline is approximated by a set of line segments, or called a polyline, the vector \( v_s \) on the nearest point between a line segment \( \{p_i, p_{i+1}\} \) of a streamline can be linearly interpolated from the vectors at \( p_i \) and \( p_{i+1} \).

For a given 3D vector field \( V(x) = \{v_x | x \in R^3, v_x \in R^3\} \), where \( x \) is a position in the vector field, and a set of streamlines, the coherence field \( C_S \) is defined by evaluating the vector coherence metric at every point \( x \). The coherence field measures the local coherence in the vector field between the true vector in the vector field and what can be inferred from the set of streamlines in its local region. Regions with low vector coherence indicates the existing streamlines can not properly represent the flows in the local regions. Therefore, by visualizing the coherence field, the user can decide where more streamlines are needed.

3.2. Streamline Placement

Volume rendering or slicing planes can be applied to render the coherence volume to provide the user with visual feedback related to the representativeness of the existing streamlines. With the information, the user can manually place more seeds to refine the visualization. If the user desires to insert streamline seeds automatically and intervene only when necessary, in this section a heuristic seeding algorithm is proposed to automatically and incrementally refine the coherence field by finding locally optimal seeds. For a given set of initial streamlines \( S \), the algorithm works as follows:

1. Generating/Updating the coherence volume.
2. Selecting \( K \) seeds with the smallest coherence
Generating streamlines from these \( K \) seeds.

(4) Evaluating the coherence of streamlines.

(5) Insert the streamlines of small coherence into the set \( S \).

(6) Repeating step 1-5 until the average vector coherence of the coherence volume satisfies a threshold \( \theta \), or the number of streamlines in the set \( S \) exceeds a user-defined maximum.

The following subsections discuss the detail of each step.

**Adaptive Seed Selection:** The basic idea of seed selection is to compare the vector coherence of each voxel with a user-defined threshold. Because of the large number of voxels in a 3D volume, it is usually too expensive to select all streamline seeds whose vector coherence are under a threshold. Therefore, only \( K \) seeds are selected in each iteration and the improvement to the average vector coherence of the coherence volume is monitored. \( K \) is increased in the next iteration if the improvement in the average coherence is below a threshold, which means the last iteration may not insert enough streamlines to improve the coherence field. Otherwise \( K \) is decreased, but \( K \) will keep in a pre-set \([K_{\text{min}}, K_{\text{max}}]\) range. The seeds are selected by scanning the whole coherence volume. For a voxel \( v \) whose vector coherence \( c_v \) is below the threshold, \( v \) is selected as a seed candidate and inserted into a sorted list with a priority key \( w_v = (1 - c_v) \). The priority key is used to collect seeds which have low vector coherences. We use a binary tree data structure to maintain the sorted seed candidates list. At the end of this process, if the number of leaf nodes in the binary tree exceeds \( K \), only the first \( K \) seeds are selected.

**Streamline Generation:** For each selected seed candidate, a streamline is generated by Runge-Kutta fourth order integrator with adaptive step sizes. Streamlines are integrated as long as possible, and they are terminated only when they integrate out of boundary, reach critical points, or generate loops. To detect self loops, streamlines are voxelized and the number of times a streamline passing through the same voxel is counted. If the number exceeds a threshold value, the streamline integration stops. All generated streamlines are stored in a k-d tree with their seed positions to save the cost of streamline generation in the future iterations. The up vectors of the streamline are also obtained and stored, which are used to define the local frame to capture the rotation of flow. An iteration index is attached on each streamline to indicate the age of the streamline.

**Coherence Evaluation of Streamlines:** To control the number of streamlines inserted into the scene, out of the \( K \) streamlines generated in each iteration, we only insert those which have an overall lowest coherence across the streamline points. This is done through coherence evaluation of streamlines, defined as follows. For each streamline \( L = \{p_i|p_i \in R^3, i \in N, i = 0..K - 1\} \) generated in the iteration, the coherence of the streamline \( c_L \) is accumulated by

\[
    c_L = \frac{\sum_{p \in L} (c_p + 1)}{2K}
\]

(2)

where \( c_p \) is the vector coherence for the streamline sample point \( p \). By comparing the coherence measures of streamlines generated in each iteration, \( K \) streamlines with the smallest coherence are selected and inserted into the set \( S \) to update the coherence volume. At each iteration, voxels passed by the streamline are marked therefore will not be selected as seeds in further iterations. Streamlines that have been generated but not selected to display in the current iteration will be considered again in the future iterations.

### 3.3. Trace Volume Generation

The streamlines generated from user input or from the seeding algorithm mentioned above are used to generate the *trace volume*, an intermediate volumetric representation for three dimensional vector fields. The trace volume stores texture coordinates which are used to look up the appearance texture to generate the final image. The concept of trace volume was first introduced in \(^6\), where trace volumes were generated by discrete voxelization of streamlines. In this section, we describe a novel method for trace volume generation. We use distance fields to generate a continuous occupancy of the volume, which provides features
such as thickness control and solid and smooth interpolation of texture coordinates. Besides, using the new representation, visual enhancements in rendering such as anti-aliasing, lighting and depth cues can all be done without extra cost.

The trace volume is a volume defined at a regular Cartesian grid, and is constructed by voxelizing streamlines and their proximity with information related to streamline parameterization. Specifically, each voxel in the trace volume contains a four-tuple vector: the Euclidean distance to the nearest streamline, the parameterized position of the nearest point on the streamline, and the UV coordinates with respect to a local frame defined at the nearest point on the streamline.

Every input streamline consists of a sequence of connected line segments. Each line segment \( s_i \) consists of two end points \([p_i, p_{i+1}]\), which are associated with parameterized 1D positions \([l_i, l_{i+1}]\), and an up(Y) vector \( u_i \). To compute \( l_i \), we have \( l_0 = 0 \) and \( l_{i+1} = l_i + m(p_{i+1} - p_i) \), where \( m \) is a value to scale the streamline length into the [0, 1] range. For each line segment \( s_i \), the up vector \( u_i \) is obtained by considering the flow rotation using a method similar to the one described in(17). With the up vector and the streamline tangent vector at each point, a local coordinate frame can be constructed.

Given a voxel, the local frame at its nearest point on the streamline is defined by the frame perpendicular to the tangent (Z) vector of the streamline, oriented by the up(Y) vector with the origin at the streamline point. For a voxel centered at position \( p \), the nearest point \( pr \) on a line segment \( s_i = [p_i, p_{i+1}] \) can be found by the distance field of \( s_i \) so that \( d(p, s_i) = |p - pr| \), where \( d(p, s_i) \) is the shortest Euclidean distance function between a point and a line segment. Therefore for a set of streamlines \( S \), the nearest line segment \( s_i \) to \( p \) is

\[
s_i = \arg \min_{s_j \in S} d(p, s_j),
\]

and the nearest point \( pr \) on \( s_i \) is solved by finding the \( t \) in \( d(p, s_i) \) where \( pr = p_i + t(p_{i+1} - p_i) \). The tangent(Z) vector \( v \) on \( pr \) is interpolated in the same way as in the vector coherence evaluation described above, and the up(Y) vector \( u \) is then interpolated by \( t \) as \( u = \frac{1 - t}{l_{i+1} - l_i} (l_{i+1} - l_i) \).

The X vector is generated by a cross product of Y and Z vector, \( x = u \times |u| \). The UV coordinate for the voxel is then calculated by

\[
UV = \left( \frac{(p - pr) \cdot x}{|x|^2}, \frac{(p - pr) \cdot u}{|u|^2} \right),
\]

the parameterized position is \( l = l_i + t(l_{i+1} - l_i) \), and the distance is \( d = d(p, s_i) \).

The voxelization is done by computing a discrete local distance field and interpolating attributes of streamlines for voxels near the streamlines. This allows a more reliable and complete voxelization of streamlines and around their proximity. Each voxel near the streamline records a precise distance to the streamline and therefore enables anti-aliasing volume rendering and thickness control.

4. Volume Rendering

Instead of rendering streamlines with lighting using line primitives, volume rendering is employed to enhance depth cues and reduce visual cluttering. Although the detail of individual streamlines may be missing due to the finite volume resolution but global effects can be better created to reveal the underlying flow patterns. Given a set of streamlines, the trace volume is constructed as described above. In the volume rendering stage, user-defined 3D solid textures can be used to illustrate the volume with various appearances. With the information stored in the trace volume voxels, streamline thickness, depth cueing, lighting, clipping, and animation can be easily controlled. Besides, the coherence volume can also be visualized to provide additional visual feedback to the user. Manual streamline seed selection can be assisted by visualizing the coherence volume.

4.1. Rendering Algorithm

Volumetric data sets can be now efficiently rendered in hardware by slices of 2D textures or view-aligned slices with 3D textures. Raycasting is also feasible using GPUs. In our
approach, we use view-aligned slices with 3D texture mapping hardware. When performing volume rendering, a set of view-aligned slices are rendered with 3D volume texture coordinates. Each fragment will sample the trace volume to get a four-tuple vector. The components in the four-tuple vector will be used as the texture coordinates to look up the appearance texture, as described in (6), as well as performing various types of rendering control. In the following, we use $x_v$, $y_v$ to represent the local frame coordinates of voxel $v$, $d_v$ as the shortest distance, and $l_v$ as the parameterized 1D position.

### 4.1.1. Thickness Control

Thickness control of streamlines during volume rendering can be easily achieved because the trace volume contains information about the distance $d_v$ from a voxel to its nearest streamline. For a user-specified thickness $T$, the opacity $\alpha_v$ of a voxel $v$ can be computed as:

$$\alpha_v = \text{clamp} \left( \frac{T}{2d_v} \right),$$

Voxels within the radius of $\frac{T}{2}$ will be rendered in opaque and voxels outside this distance range will fade to transparent. $t$ is a parameter to decide the speed of fading. With a smooth transition from fully opaque to fully transparent, anti-aliasing on the boundary of streamlines can be achieved.

By controlling the thickness of streamlines, it is possible to add additional depth cues to the final images. For example, to enhance depth perception without lighting, dark colors on the streamline boundary or the region outside the streamline can be used to create a haloing effect. To achieve this, the output color $c_v$ is modified by $\alpha_v$ as $c_v' = \alpha_v c_v$. Figure 2 shows streamlines with gray strip patterns generated from a Solar plume data set.

### 4.1.2. Appearance Control

For a given user-defined appearance volume texture, the color of the voxel can be sampled by the 3D texture coordinates $(\frac{x_v}{T} + 0.5, \frac{y_v}{T} + 0.5, l_v)$ where $T$ is the user-controlled thickness. The appearance texture is used to control the shape and color of the rendered streamlines. Using a proper appearance texture can help the user distinguish streamlines, local flow direction, and the twist of streamlines. Figure 3 shows results using colored lines appearance texture and a noise volume texture to visualize the plume data set. To animate the streamlines with the appearance texture, $l_v$ can be increased by an offset at every frame. Figure 4 shows the use of volume appearance texture for a Tornado data set to indicate flow directions with arrows, contrast colors and particle-like effects. They also can be easily animated.
Fig. 3 The enhancement of visual effect using appearance volume textures comparing to (a) rendering lines with lighting: (b) using colorful lines pattern to indicate the twist of flows and using (c) noise volume texture to differentiate streamlines and imitate flows in animation.

Fig. 4 The shape and color of volume appearance texture are useful in animation to indicate (a) flow direction by arrows, (b) flow movement by contrast colors, and (c) particle-like effect. (a) and (b) are also rendered with lighting.

4.1.3. Lighting Generally speaking, lighting in volume rendering usually requires pre-computed volume normals. However, in our approach, the normals of the voxels can be directly derived from the gradients of the distance field to the streamlines. The gradient $g_v$ of the voxel $v$ can be generated by central difference of the distance in three axes:

$$g_v = (d_{v+x} - d_{v-x}, d_{v+y} - d_{v-y}, d_{v+z} - d_{v-z}),$$

where $dx$, $dy$, and $dz$ are the distances between discrete voxels in the three axes of the texture coordinate space. Then the normal $N_v$ of voxel $v$ is obtained by normalizing $g_v$.

With the normalized vectors from voxel $v$ to the light source, $L_v$, and to the eye, $E_v$, the color $C_v$ can be calculated by the Phong illumination model for diffuse and specular components:

$$C_v = C_{diff}(N_v \cdot L_v) + C_{spec}(ref(-L_v, N_v) \cdot E_v)^n,$$

where $C_{diff}$ is the diffuse material color sampled from the appearance texture, $C_{spec}$ is the specular material color which is usually the white color, and $n$ is the specular exponential value. $ref(i, n)$ is a reflection function where $ref(i, n) = i - 2n(i \cdot n)$. With lighting on the volumetric streamlines, the 3D appearance of streamlines can be clearly presented.

4.2. Coherence Volume Rendering with Manual Streamline Seed Selection

Since the coherence volume is a scalar volume, it can be easily visualized with volume rendering. A transfer function is used to map the vector coherence $c$ to a color value (RGB):

$$R = \frac{1}{1+c^2}, \quad G = \frac{c}{1+c^2}, \quad B = Scale(d),$$

where $d$ is the distance to the nearest streamline, $Scale(d)$ scales $d$ into the range of [0,1]. With the transfer function, regions of low vector coherence will be in red to pink depending
on the distance value, and regions of high vector coherence between green and blue. To assist the users manually selecting seeds, we allow them to slice through the coherence volume and select seeds directly from the screen. The seed positions are obtained by computing the intersection between the ray from the screen pixel and the coherence volume slice. Figure 5 shows an example of the coherence volume slicing during manual streamline seed selection. When a streamline is generated, the coherence volume will be updated to reflect the new streamline set.

When rendering the coherence volume, the alpha value of the voxel can be set to $k - c^2$, where $k \in [0, 1]$ is used to control the opacity of the volume, and allow regions of low coherence to be more opaque than regions of high coherence. The trace volume can also be merged with the coherence volume to demonstrate the correlation between the trace volume and the coherence volume, which indicates the representativeness of the trace volume. To combine two volumes, the output color $Color_v$ of a voxel $v$ is interpolated by

$$Color_{v,RGB} = C_v.A + C_v.RGB + (1 - C_v.A)T_v.RGB.$$  
$$Color_{v,A} = C_v.A + T_v.A.$$  

Where $C_v$ is the color of the coherence volume and $T_v$ is the color of the trace volume at voxel $v$. In this way, the volumetric streamlines will be rendered with the color of trace volume and regions of low coherence will be rendered in red. By adjusting the thickness of the streamlines, the streamlines rendering can be turned on or off in the coherence volume rendering. Figure 5 shows an example of the coherence volume and the combined volume rendering.

5. Experimental Results

We implemented the proposed algorithm using shader model 3.0 with HLSL and DirectX 9.0c. The program was running in a Core Duo 1.86 GHz laptop with 2 Gigabytes of memory and an nVidia GeForce 7600 Go GPU. Two 3D vector data sets were used to evaluate the performance of our algorithm, including coherence volume generation, trace volume generation and automatic streamline seeding.

Table 1 lists the performance of coherence volume generation and automatic streamline seeding selection. Stop threshold of vector coherence is the cosine of tolerant angle $\theta$. Active seed candidates in each iteration were bound in the range of [1000, 3000], where the unsatisfied seeds may usually exceed 500K between iterations. The initial streamline set is given by choosing a streamline passing through the central region of the vector field. The number of iterations in the seeding algorithm is the number of streamlines in the final set minus one. The time needed for coherence volume generation is proportional to the size of the coherence volume. The higher vector coherence threshold ($\cos(\theta)$), the more time in process, and more streamlines will also be generated.

Table 2 shows the performance of streamline voxelization for different resolutions of trace volumes. 20, 40, 60 streamlines are randomly generated and voxelized. Our imple-
Table 1 Performance (in seconds) of coherence volume generation and the heuristic streamline seeding algorithm.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Volume</th>
<th>Time(s)</th>
<th>#L.</th>
<th>#Seg.</th>
<th>θ (°)</th>
<th>Total(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tornado</td>
<td>96³</td>
<td>1.284</td>
<td>8</td>
<td></td>
<td>2034</td>
<td>40</td>
</tr>
<tr>
<td>Tornado</td>
<td>96³</td>
<td>1.222</td>
<td>15</td>
<td>3616</td>
<td>30</td>
<td>81.89</td>
</tr>
<tr>
<td>Tornado</td>
<td>96³</td>
<td>1.137</td>
<td>37</td>
<td>8720</td>
<td>20</td>
<td>267.84</td>
</tr>
<tr>
<td>Plume</td>
<td>63³ x256</td>
<td>2.682</td>
<td>12</td>
<td>6752</td>
<td>40</td>
<td>66.13</td>
</tr>
<tr>
<td>Plume</td>
<td>63³ x256</td>
<td>2.795</td>
<td>27</td>
<td>11964</td>
<td>30</td>
<td>147.30</td>
</tr>
<tr>
<td>Plume</td>
<td>63³ x256</td>
<td>3.762</td>
<td>72</td>
<td>32460</td>
<td>20</td>
<td>477.45</td>
</tr>
<tr>
<td>Plume</td>
<td>126³ x512</td>
<td>14.150</td>
<td>11</td>
<td>10404</td>
<td>40</td>
<td>211.42</td>
</tr>
<tr>
<td>Plume</td>
<td>126³ x512</td>
<td>15.073</td>
<td>21</td>
<td>18727</td>
<td>30</td>
<td>434.91</td>
</tr>
<tr>
<td>Plume</td>
<td>126³ x512</td>
<td>14.351</td>
<td>55</td>
<td>36888</td>
<td>20</td>
<td>1186.74</td>
</tr>
</tbody>
</table>

#L. = #Lines. 1 Coherence volume generation per iteration. 
#Seg. = #Segments. 2 Total time cost of the seeding algorithm.

Table 2 Performance (in seconds) of trace volume generation for randomly generated streamlines (20, 40, 60 lines) at different volume resolutions.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Vector Field</th>
<th>Volume</th>
<th>#20L(s)</th>
<th>#40L(s)</th>
<th>#60L(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tornado</td>
<td>96³</td>
<td>96³</td>
<td>0.19</td>
<td>0.36</td>
<td>0.51</td>
</tr>
<tr>
<td>Tornado</td>
<td>96³</td>
<td>192³</td>
<td>0.82</td>
<td>1.49</td>
<td>2.12</td>
</tr>
<tr>
<td>Tornado</td>
<td>96³</td>
<td>384³</td>
<td>5.34</td>
<td>9.73</td>
<td>14.02</td>
</tr>
<tr>
<td>Plume</td>
<td>63³ x256</td>
<td>63³ x256</td>
<td>0.28</td>
<td>0.43</td>
<td>0.60</td>
</tr>
<tr>
<td>Plume</td>
<td>63³ x256</td>
<td>126³ x512</td>
<td>1.09</td>
<td>1.68</td>
<td>2.35</td>
</tr>
<tr>
<td>Plume</td>
<td>126³ x512</td>
<td>126³ x512</td>
<td>0.74</td>
<td>0.94</td>
<td>1.27</td>
</tr>
</tbody>
</table>

Fig. 6 Image quality comparison on an streamline in the volume rendering with volume textures at different resolutions.

In our experiment, the average vector coherence was improved quickly and reached 0.8(cos(36°)) within 1/3 of the whole iterations. Then the improvement of the average vector coherence slowed down in the rest of iterations. Although the average vector coherence was high, some seeds at a few spots of low coherence may still not be selected because of their accumulated coherence errors along the streamlines are not the maximum among candidate seeds. Those local points can be easily detected by visualizing the coherence slice or coherence volume. Manual seed selection by visualizing the coherence volume is helpful. Once a...
seed is manually selected, the streamline and coherence volume is updated. And the change of the vector coherence in both coherence volume and slice is noticeable. Although critical points are not easily classified by looking at the coherence volume/slice, major vortices and turbulence between regions are detectable by eyes because the vectors in those regions are usually quite different from the nearby regions. Overall, the seeding algorithm can capture the overview of the flow with few representative streamlines. For example, as shown in Figure 7, a macro visualization of the flow pattern of the tornado can be characterized by 8 streamlines. Figure 8 shows an example of visualization with only 12 streamlines.

Using the coherence field the problem of streamline seed selection can be reduced to a coherence maximization problem. However, the optimal solution is difficult to search because there exist an infinite number of candidate streamlines in the vector field. The number of seed candidates in each iteration will affect the number of resulting streamlines selected for a given vector coherence threshold. The initial set of the streamline will also influence the final set of streamlines but will not seriously affect the final visualization quality. A proper initial
streamline set such as generated from templates around critical points would benefit the seed selection.

6. Conclusion

We have presented an end-to-end volume rendering framework for visualizing 3D vector fields. An intermediate representation of the vector field, called trace volume, is proposed to allow a flexible control of texture appearance and a variety of depth cueing techniques. The concept of coherence field is introduced to evaluate the similarity between the directions of existing streamlines and the actual vectors in the vector field. The coherence values, in the form of coherence volume, can be either displayed to the user as additional visual feedback to guide seed selections, or utilized by a heuristic seeding algorithm to generate the streamlines. The proposed seeding algorithm can capture the global structures of the vector field for effective overview of the data.

In the future, we would like to extend our framework to visualizing time-varying vector fields. We will also evaluate our heuristic streamline seeding algorithm with more scientific data sets. Finally, transfer function design will be a focus of our future work for rendering the trace volume.

References


