High-Quality and Interactive Animations of 3D Time-Varying Vector Fields

Anders Helgeland, *Member*, *IEEE*, and Thomas Elboth

Abstract—In this paper, we present an interactive texture-based method for visualizing three-dimensional unsteady vector fields. The visualization method uses a sparse and global representation of the flow, such that it does not suffer from the same perceptual issues as is the case for visualizing dense representations. The animation is made by injecting a collection of particles evenly distributed throughout the physical domain. These particles are then tracked along their path lines. At each time step, these particles are used as seed points to generate field lines using any vector field such as the velocity field or vorticity field. In this way, the animation shows the advection of particles while each frame in the animation shows the instantaneous vector field. In order to maintain a coherent particle density and to avoid clustering as time passes, we have developed a novel particle advection strategy which produces approximately evenly-spaced field lines at each time step. To improve rendering performance, we decouple the rendering stage from the preceding stages of the visualization method. This allows interactive exploration of multiple fields simultaneously, which sets the stage for a more complete analysis of the flow field. The final display is rendered using texture-based direct volume rendering.

Index Terms—Vector field visualization, flow animation, direct volume rendering, multifield visualization, programmable graphics hardware.

1 Introduction

As a consequence of our increased ability to model and measure a wide variety of physical phenomena, we are overwhelmed with data. This is especially true in the world of unsteady fluid mechanics and other disciplines involving time series of vector data. Analysis of fluid dynamics imposes significant requirements on the visualization system and demands interactive navigation of the vector data to achieve a proper investigation of the flow. A number of sophisticated visualization techniques have been proposed in the past to assist scientists in the analysis of flow fields. These techniques can be loosely classified into geometry-based and texture-based methods. Geometry-based methods use graphical primitives to convey the local behavior of the flow. In both steady and unsteady flows, these primitives originate from a set of user defined seed points. Examples of such visualization objects are points, glyphs, field lines, path lines, stream surfaces [1] and flow volumes [2] to name a few. While the geometry-based techniques can effectively visualize local flow features, texture-based methods attempt to create a continuous visual representation to illustrate the global behavior of the flow [3], [4], [5], [6], [7], [8]. In two dimensions, texture-based methods are capable of offering a clear perception of the entire vector field. However, in three dimensions, it can be challenging to find a good visual representation of the flow due to perceptional difficulties encountered with dense 3D displays such as occlusion and cluttering. Perception can be improved by highlighting important features of the flow and by using sparse 3D textures [9], [10]. However, by using such an approach we encounter the same challenge as for any particle-tracing method, which is choosing appropriate seed points for the particle tracing in order to visualize all important features of the flow.

This paper presents a novel texture-based method for visualizing 3D unsteady flow fields based on a sparse representation of the time-dependent vector fields. In this method, an even distribution of particles covering the entire physical domain are tracked along the time-dependent velocity field. To obtain a global representation of the flow, we maintain the density of particles in the domain at all times. This is done by injecting and removing particles in areas with too low or too high density, respectively. By doing this, we are in fact emulating a field description rather than a true particle description of the flow. Hence, we provide a method that gives a global representation of the flow, but does not suffer from the same perceptual issues as is the case for visualizing dense representations. At each time step, directional information are embodied in the output 3D texture. Thus, the animation shows the advection of particles while each frame in the animation shows the instantaneous vector field, which can be any vector field including the velocity and the vorticity field.

To improve rendering performance we have, as the method proposed by Li et al. [11], decoupled the rendering stage completely from the rest of the visualization pipeline. Hence, the final rendering of the flow data will achieve an increase in performance compared to the combined advection and rendering algorithms. Consequently, we can afford doing some interesting multifield visualizations.

2 Related Work

Several texture-based techniques have been proposed to produce a dense and global representation of unsteady

- *A. Helgeland is with the University of Oslo and the University Graduate Center, Unik, PO Box 70, N-2027 Kjeller, Norway. E-mail: andershe@ifi.uio.no.*
- *T. Elboth is with the University of Oslo and the Fugro Geoteam, Hoffsveien 1C, PO Box 490, N-0213 Oslo, Norway. E-mail: thomae@math.uio.no.*

Manuscript received 3 Nov. 2005; revised 23 Feb. 2006; accepted 14 Mar. 2006; published online 8 Sept. 2006.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-0149-1105.

vector fields. Early techniques include the method proposed by Forssell [12], and the UFLIC method proposed by Shen and Kao [13]. Both these methods are based on the Line Integral Convolution (LIC) technique.

A related approach makes use of texture advection. Examples of texture advection techniques are the method proposed by Max and Becker [14], the Lagrangian-Eulerian Advection (LEA) methods [6], [15], [16], and the Image-Based Flow Visualization (IBFV) technique [7]. Although most of these techniques are restricted to 2D flows, some work has been done to also incorporate 3D flows [17], [18].

Visualization of 3D flows using dense texture-based techniques is challenging due to perceptual difficulties such as occlusion and cluttering. Dense volume textures can be more effectively represented by highlighting important features of the flow and by using sparse instead of completely dense 3D textures [9], [10]. Perception of 3D flows can also be improved by interactive user interventions such as clip planes [19] and by the use of halo and shading techniques to enhance depth perception [9], [10], [20].

Some previous work exists which is also capable of producing a global but sparse representation of steady 3D vector fields [21], [11]. Such approaches reduce the clutter and occlusion problems inherent for dense texture-based methods. Guthe et al. proposed an adaptive distribution of animated particles to visualize steady 3D flows [21]. While the method by Li et al. [11] represents the 3D flows as a collection of densely placed stream lines. The latter method has been extended to also handle time-varying vector fields [22]. Here, the 3D flows are represented as a dense set of path lines.

Two techniques of particular interest in the context of this paper are Dynamical LIC (DLIC) [23] and Unsteady Flow Advection-Convolution (UFAC) [24]. DLIC is an extension of LIC to generate animations of fields in which the motion of the field lines is determined by a second vector field. The technique is specifically developed for visualizing time-dependent electromagnetic fields. Weiskopf et al. [24] adopts this idea into the UFAC method. Their method provides a spacetime-coherent dense representation of time-dependent vector fields using a similar two-step process. Here, trajectories along path lines are used to obtain temporal coherence, while convolution along stream lines are used to obtain spatially correlated patterns. Originally, the UFAC method was based on texture-based backward gathering, but a new and improved implementation of their framework replaces the backward gathering with particle-based forward integration [25]. It should be noted that both DLIC and UFAC are restricted to time-dependent 2D vector fields.

3 Algorithm Overview

To generate high quality and interactive 3D animations, we have developed a novel time-varying algorithm for visualizing three-dimensional vector fields. The 3D animation is made by injecting a collection of "evenly distributed particles"[1] throughout the whole domain. These particles are then tracked along the time-dependent velocity field by calculating their path lines. In order to maintain a coherent particle density and to avoid clustering as time increases, we have developed a novel particle advection strategy. This strategy is inspired by an algorithm used to visualize evenly-spaced stream lines in 2D steady flows [26]. At each time step, these particles are used as seed points to generate field lines using a 3D texture-based method such as the Seed LIC method [10]. In this way, the animation shows the advection of particles, while each frame in the animation shows the instantaneous vector field. Our hybrid solution, based on both the use of path lines and field lines, is based on an idea very similar to the ones used in DLIC and UFAC.

The time-dependent vector field visualization algorithm is divided into three parts:

1. Particle Advection—preprocess (Section 4).
2. Field Line Generation—preprocess (Section 5).
3. Volume Rendering (Section 8).

The first two stages of the visualization pipeline are decoupled completely from the rendering stage of the pipeline, and are considered as a preprocess. This ensures a minimum of computation in the rendering stage and, hence, enables interactive visualizations. Since the GPU calculations are kept to a minimum during the rendering, we can also afford doing some interesting multifield visualizations without loosing interactivity.

4 The Particle Advection Strategy

One of the main challenges encountered when visualizing three-dimensional vector fields is finding a good visual representation of the vector field due to perceptual issues such as occlusion and cluttering. This is especially true for dense texture-based representations of 3D flows such as the LEA and the IBFV methods. Visual perception can be improved by replacing the completely dense input noise texture by a sparse input texture [9], [10]. The sparsely collected points will act as seed points with the result of rendering a collection of densely placed field lines instead of a more or less solid object as is the case for a dense representation.

The fundamental problem with the sparse texture approach is to choose appropriate seed points for the particle tracing of the flow. Such a particle advection method needs to address a number of issues: It must be able to depict all important features of a flow, maintain a certain particle density both in time and space, properly take into account certain boundary effects such as in and out flow, and make sure that the field lines traced from these seed points remain at a certain distance to each other. All these issues are addressed in the particle advection algorithm proposed in this paper. A flowchart of the algorithm is shown in Fig. 1.

4.1 Initial Seed Point Placement

Our goal of visualizing vector fields is to depict all interesting parts of the flow, while at the same time avoiding issues like clutter and occlusion. One way to improve the understanding of the flow field is to reduce cluttering by making sure that the field lines are separated by a certain distance to each other. The algorithm designed to achieve this begins with an initial

1. A more precise description of "evenly distributed particles" is given in the second paragraph of Section 4.1.

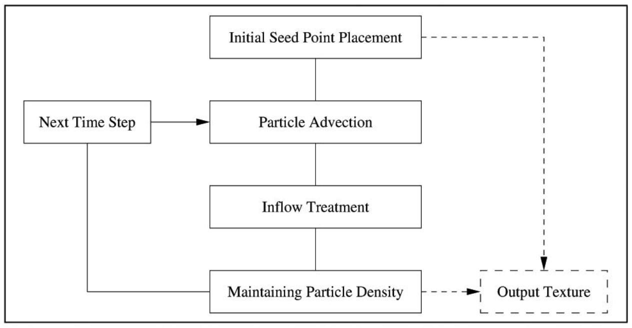

Fig. 1. Flowchart of the particle advection algorithm.

seeding strategy inspired by an algorithm by Jobard and Lefer used to visualize evenly-spaced stream lines in 2D steady flows [26].

The purpose of the seed point placement algorithm is to obtain a collection of evenly distributed seed points throughout the domain which meet two criteria. First, the chosen seed points should be separated in such a way that all field lines traced from these seed points are separated by a certain distance. Second, the distribution of points should be somewhat random in order to avoid a completely uniform distribution which produces visual artifacts.

The initial seeding algorithm starts by inserting a seed point randomly into the domain. Then, a field line is computed in upstream and downstream direction given a user-defined length. If any sample point along the computed field line has a less distance to any other already inserted field lines than a prescribed minimum separating distance, the particle is rejected. If not, the particle is inserted into the domain and all cells or voxels covered by the field line are marked. A single sample point is checked for validation by checking neigboring voxels for marked values. The minimum separating distance allowed, d_i, is given in number of voxels. The algorithm can be described by the pseudocode given in Fig. 2. One major difference between our algorithm and the proposed method by Jobard and Lefer, is that our output is a point set represented as a sparse 3D texture instead of a field lines set. Also, the same fixed field line length is used in all field line computations. The final set of chosen seed points defines the sparse particle texture and are set to 255, which is the maximum value for a byte texture (of type unsigned char). The remaining voxels are set to zero. This texture will, at a later stage of the algorithm, be treated as input to a texture-based method for representing directional information via patterns of correlation in a texture. Here, the same instantaneous vector field and field line length will be used. This is covered in Section 5. All field line computations are done using a fourth-order Runge Kutta method. In addition to the particle texture, all particles are stored in a separate list. This list is used when traversing through the particles during the advection step presented in the following section.

4.2 Particle Advection

After the initial placement of the seed points, these points are treated as particles and are advected along the velocity field for a short time period Δt. The path lines are also computed using a fourth-order Runge Kutta method. To achieve correct discrete Runge Kutta integration, intermediate steps are calculated from interpolated vector field values. This is done by linear interpolation. To avoid

```
for each voxel $v$ (in random order)
    compute field line from center of $v$ in both directions
    if (some point segment on the field line falls too close
        to an already inserted field line) then
            reject seed point
    else
            mark all voxels covered by the field line
            insert seed point
```

Fig. 2. Pseudocode of the initial seed point placement algorithm.

cluttering, all particles that come too close to any other particle are removed after each time step. When particles are inserted into the domain, the separating distance between adjacent particles apply for all sample points along the field lines traced from these particles. However when removing particles, we have found that it is sufficient to compute the separating distance just between the particles. In addition, to avoid that too many particles are removed during each iteration, particles are allowed to come closer to adjacent particles than the initial separating distance allows. We have found that choosing the removing separating distance, d_r, to be one half of the inserting separating distance, d_i, gives good results. This may of course vary for different data sets.

4.3 Inflow Treatment

While all particles leaving the physical domain are removed naturally in the particle advection step, special attention has to be given to particles entering the domain. After each advection step, every cell or voxel on the boundary are checked for inflow in random order. The inflow particles are then inserted into the domain using the same criteria as was used for the initial seed point placement. Since the particles are inserted at the boundary, only the downstream direction of the field lines need to be computed. The algorithm for inflow treatment is given in Fig. 3. For efficiency, all voxels on the boundary are stored in a separate list.

4.4 Maintaining Particle Density in Time and Space

As time goes and the particles start to cluster, some areas of the domain will have lower density compared to the initial distribution. To maintain an even distribution in space, we therefore inject particles in areas with low density. This ensures that all parts of the flow are represented at all times. By doing this, we are in fact emulating a field description rather than a true particle description of the flow. The injected points are chosen randomly according to the algorithm in Fig. 2. In order to maintain the density of particles in time, a

```
for each voxel $v$ on boundary (in random order)
    if (inflow)
        compute field line from center of $v$ (downstream)
        if (some point segment on the field line falls too close
            to an already inserted field line) then
                reject seed point
        else
                mark all voxels covered by the field line
                insert seed point
```

Fig. 3. Pseudocode of the inflow treatment.

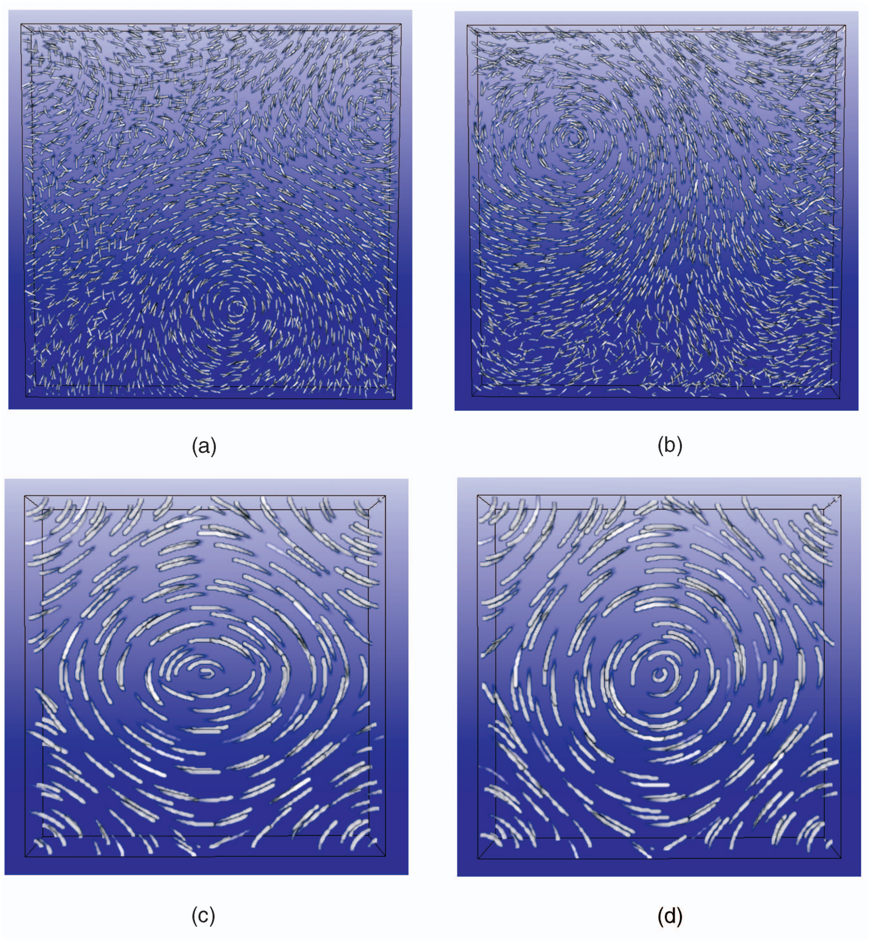

Fig. 4. Illustration of the distribution strategy for two different data sets. The figures show the distribution of particles and field lines for the first and final time step of the hurricane Isabel and a data set generated from a Weather Research and Forecasting (WRF) model. We see from the images that the initial even distribution of field lines is well preserved during the animation. (a) Isabel ($t = 0$), (b) Isabel ($t = 48$), (c) WRF ($t = 0$), and (d) WRF ($t = 80$).

max number of seed points, *MAX_NUM_SP*, allowed in the domain is computed at the beginning of the algorithm. The *MAX_NUM_SP* is defined by the initial seed point placement algorithm as the number of starting particles. Only when particles are removed, either due to outflow or clustering, new particles can be injected into the domain.

The method by Guthe et al. [21] also supports the injection and removal of particles to maintain a certain distribution. In their method, particles are initially distributed using a weight function allowing particle density to vary locally according to regions of interest in the volume. The injection and removal of particles are dependent on both how the density is preserved in these regions and on the age property attached to each particle. The weight function allows a combination of various physical quantities in addition to the local gradient, divergence and curvature of the vector field to influence the particle distribution. This allows visualization of vector field features without the need of extracting them. Focusing on a region of interest also diminishes the occlusion effect.

We use an alternative approach to address the perceptual difficulties related to volume visualization and how to focus on various volume data features. Our distribution method only controls the global density of the particles distributed into the system. This means that an even distribution of particles is preserved at all times. Once the advection and field line generation steps of the visualization pipeline are finished, the result can be sent to our volume rendering framework for analysis. Here, regions of interest can be emphasized using multifield visualization techniques and by manipulating transfer functions interactively (Section 8). In addition, interactive clip planes and selection of subsets can be used to facilitate the investigation of the volume data. This will further reduce the occlusion effects. We believe that our framework provides a more flexible solution compared to the solution proposed by Guthe et al. [21]. In our framework, different physical quantities are used only to determine the opacity value of each voxel and, thus, controlling the regions of interest in the final rendering stage, and not during the particle distribution stage of the algorithm. Separate volume data features can also be rendered together with directional information of the vector field when using the rendering framework presented in Section 8. Another advantage with our approach, is that all field lines in each frame in the animation are more or less separated by a minimum distance to each other. Such a solution gives very easy to interpret still images (volumes) and is nearly optimal for diminishing cluttering problems.

Fig. 4 shows the distribution of particles after the first and final time step for two different data sets. In Fig. 4a and

Fig. 4c, it is clear that all particles and their field lines are evenly placed throughout the domain after the first time step. Since some clustering of particles may happen as time passes, the separating distance between adjacent field lines will vary some during the animation. However, since particles are injected in areas with low density, an even distribution in space at all times is preserved. This can be seen in Fig. 4b and Fig. 4d.

4.5 Subsampling

The algorithms described in Fig. 2 and Fig. 3 can gain speed by picking only a subsample of all cells or voxels in the domain and at the boundary. To ensure an even distribution of particles, all subsamples are taken uniformly. The subsample is done by traversing through the cells in the domain using a step size larger than one. For each sample we store the cell coordinates (i, j, k) in a new array. This new array will then be used by the algorithms in Fig. 2 and Fig. 3. Thus, new particles can only be inserted into the domain from these chosen subcells. The distance between the samples should always be smaller than the inserting separating distance d_i.

Choosing a subsample will result in fewer particles distributed into the system. An average of approximately 10 percent fewer points was found when testing the algorithm using a step size of two in each directions (see Section 9).

5 Field Line Generation

After each time step in the particle advection algorithm, the resulting sparse 3D texture can be used as input to a texture-based algorithm for visualizing field lines. Here, any instantaneous vector field can be used including the velocity field and the vorticity field. This is not possible with the LEA and IBFV methods, since the spatial correlation in each image is tied directly to the velocity field following path lines. Another advantage with our approach is that each frame in the animation reveals the true instantaneous direction of the flow, and not just an approximation of it in form of short path lines.

We have tried three different approaches for generating field lines. These are Seed LIC, anisotropic diffusion, and a direct approach.

5.1 Seed LIC and the Direct Approach

The first method we tried was the Seed LIC method by Helgeland and Andreassen [10]. Seed LIC is a fast method for computing 3D LIC textures and is based on the Fast LIC method by Stalling and Hege [27] and the work done by Interrante and Grosch [9]. The technique exploits the sparsity of the input texture by calculating field lines and computing the convolution starting from the seed points only. This results in a 3D texture where the voxel values along the individual field lines are highly correlated. However, when using Seed LIC on input textures such as the ones produces by the algorithm described in Section 4, all individual field lines in the final image will be covered by voxels with more or less the same intensity value. The reason is that, in most cases, only the starting voxel (seed point) will make a positive contribution to the convolution integral. This means that a lot of unnecesarry computations are performed.

Instead, we use a more *direct approach* when generating the field lines. In this method, all voxel values are set directly during the field line integration step, meaning that no convolution integral is computed at all. This leads to a significant speed up of the algorithm.

5.1.1 *Encoding Orientation*

To incorporate orientational as well as directional information in each output 3D texture, each voxel value along the field line in upstream direction is set with decreasing intensity values. This will have the same effect as the OLIC method proposed by Wegenkittl et al. [28]. While all voxels along field lines in downstream direction are set to 255, the upstream voxel intensity I at $\mathbf{x}_i = \sigma(s_i)$ is computed by the formula

$$I(\mathbf{x}_i) = \big((M - i)/M\big) * 255.$$

Here, M denotes the max number of samples along the field line in each direction. The curve $\sigma(s)$ is parameterized by the arc-length s.

5.1.2 *Shading*

To convey the 3D shape and depth relations among the field lines, we employ the *limb darkening* technique used in [10]. Limb darkening creates a halo effect around each field line and it is obtained by manipulating transfer functions (TFs). Since all TFs are handled by the graphics card and only require a 1D texture stored in memory, this is an efficient method for shading. Limb darkening is achieved by assigning darker values and decreasing opacity near the edges of a volume feature. To emphasize the halo effect, the output textures are convolved with an isotropic $3 \times 3 \times 3$ filter [10]. This leads to a smearing of the field lines, resulting in a smoother representation. This extra convolution step will also happen during the preprocess stage of the visualization pipeline and will not effect the rendering performance. The limb darkening technique is illustrated in Fig. 4, where it has been used to create halos around field lines generated by the direct approach. Notice also that the orientation of the flow field has been encoded into the final image.

5.2 Anisotropic Diffusion

As an alternative approach to generating field lines, we have experimented with a technique called *anisotropic diffusion* [5]. The technique is based on the idea of Line Integral Convolution (LIC), where blurring an input texture along field lines leads to an image that reveals the structure of a vector field. As oppose to LIC, which can be thought of as solving the heat equation in 1D (along field lines), anisotropic diffusion takes effect in all directions. In addition to a strong blurring effect along field lines, we have encoded a small blurring effect in the orthogonal directions. Such a solution produces field lines with very low "edge" values, while maintaining the initially higher "inner" values. This fits very well with the limb darkening approach.

Anisotropic diffusion creates very smooth and nice looking "cigar shaped" field lines without doing the extra convolution that was needed for the direct approach. An example of these "cigar shaped" field lines can be seen in Fig. 5. Even though this technique could be seen as an ideal solution, the method is unfortunately extremely computationally intensive. When used on the Isabel data, our implementation of anisotropic diffusion took about 30 minutes to compute each frame of the animation. This resulted in a total computational time of 24 hours compared to 79.2 seconds for the direct approach. As a result, we prefer using the direct approach.

5.3 Our Approach versus DLIC and UFAC

Both DLIC [23] and UFAC [24],[25] use a similar approach to ours when creating instantaneous structures at each time step in their animations. However, there are some notable differences in the way they are created. The first difference is in how the input texture prior to the convolution stages of the three algorithms is computed. In our approach, the input texture is a "pure" particle texture. That is, each cell containing a particle is set to the value 255. In the other two methods, the input texture generation is a bit more complex. Generally, here, more than one single particle is used to determine the value of each cell of the texture. Further more, when computing the convolution integral in those methods to create spatial structures in each frame, neighboring texture values along the instantaneous curves will also make a contribution to the final texture value. This means that a set of different particles and trajectories are used to create a single "structure" in each frame. Generally, in our approach, only a single particle contributes in the generation of a single field line. This is due to our distribution method that injects new particles in such a way that no other particles are allowed to come nearer any other instantaneous field line than a prescribed minimum distance. In fact, when using the direct field line generation approach presented in Section 5.1, no convolution integral is computed at all. Here, the instantaneous field lines are created directly during the integration step.

In order to elucidate how our approach differs from the DLIC and UFAC methods, we can regard the instantaneous curves in our solution as advanced glyphs attached to each individual particle. This implies that the first two stages of our visualization pipeline could just as well be implemented as a geometry-based technique.

As indicated above, the actual distribution methods used in our solution also differs from the methods used in the two other frameworks. The proposed particle distribution method in this paper is designed to give a global and sparse description of the instantaneous vector field at each instant of time. This is achieved by generating evenly-spaced field lines at each frame in the animation. The DLIC method creates a completely dense distribution by limiting each pixel to contain a maximum of 2 and a minimum of 0.5 particles. A similar explicit distribution method is also supported by the framework in [25], as this framework can be used to reproduce visualization results generated by a series of existing techniques, including the DLIC method. In addition, the very same framework also support a probabilistic distribution method to control particle density. Just as in the method by Guthe et al. [21], the injection and removal of particles can here be controlled by taking into account the divergence of the vector field. The probabilistic distribution method can be used to generate sparse representations, but it is not able to generate spatial curves that are evenly placed throughout the domain.

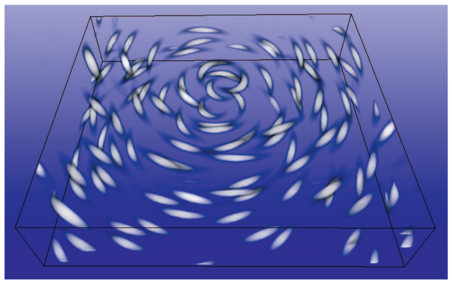
Fig. 5. Visualization of field lines using anisotropic diffusion and limb darkening.

6 Increasing Temporal Coherence

In order to make the animation smooth, it is necessary to achieve a high degree of coherence between the animation frames. In our algorithm, temporal coherence is obtained from the advection of particles, while spatial coherence is introduced by the generation of field lines traced from the advected particles at each time step. Such an approach could lead to incoherent animations, but the degree of smoothness can easily be controlled by adjusting two parameters, the first being the length of the field lines and the second being the global max distance a single particle can travel between two frames. The second parameter is controlled by the time step size Δt. Since the field lines in fact are instantaneous objects, the direction of the field lines could change rapidly for unsteady flows. In addition, a single particle that travels too far can further increase the lack of temporal coherence. Reducing either one or both of these two parameters will improve the smoothness of the animation. Even though a large Δt could cause lack in temporal coherence, spatial coherence in each frame will not be affected by the choice of Δt as is the case for the LEA an IBFV methods. In both these methods, choosing a too large Δt will lead to uncorrelated path lines.

Usually, the vector fields to be visualized are derived from a simulation as a set of N instantaneous vector fields. Whenever a smaller Δt is chosen compared to the time interval between the instantaneous vector fields, linear interpolation between successive vector fields is performed to give physically correct rendering. For the data set shown in Fig. 4a, Fig. 4b, and Fig. 6, a sufficiently smooth animation was obtained by choosing Δt to be one half compared to the time interval between two successive vector fields.

7 Physical Interpretation

The animation technique presented in this paper uses a hybrid solution based on both the use of path lines and field lines. The physical interpretation of the animation is however not necessarily straightforward, and therefore deserves some comments. Path lines are used to track individual particles over time, while field lines are used to give an instantaneous representation of the time-dependent

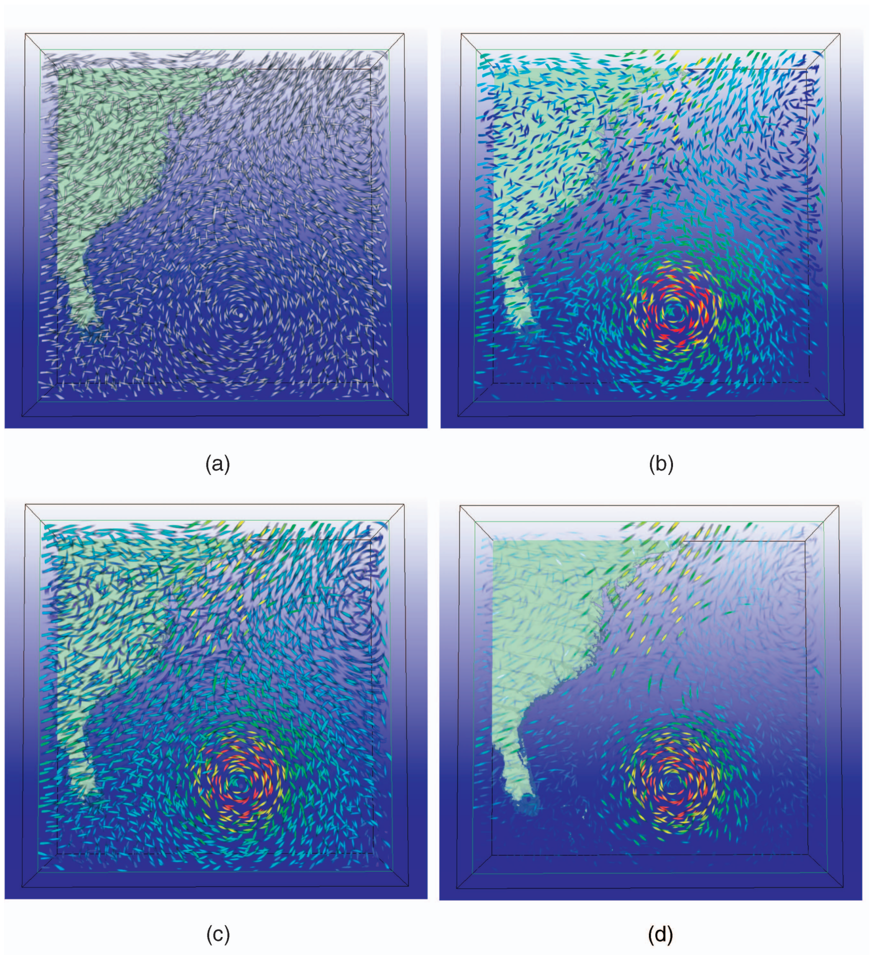

Fig. 6. Illustration of different visualization scenes used on the hurricane Isabel data set. Each scene calls a specific shader program which defines how the various volumes are blended. The ground data is in all scenes rendered as a separate volume and causes the extra L in all the shader programs (see Section 8.1). (a) **LL**-scene. Velocity (PFL volume) and ground field. (b) **(LA)L**-scene. (L,A) = (Velocity magnitude, PFL volume). (c) **L(LA)L**-scene. A scene where both scenes from Fig. 6a and Fig. 6b are blended together. Such a scene shows both the vector direction and vector magnitude in addition to depth cues in form of shading using limb darkening. (d) **L(LA)ML**-scene. Here, the first two volumes in the scene from Fig. 6c have been masked by the velocity magnitude. The masking volume has been used to emphasize the region of interest, in this case, the hurricane center.

vector field at each instant of time. One may argue that a solution based on field lines will not give an accurate description of a time-dependent vector field. The argument is based on the notion that stream lines and path lines are different in those cases. It is therefore important to note that the curves displayed in each image of the animation do not represent particle paths in general. The objective with this particular visualization technique was, however, not just to show the history of individual particles, but also to display instantaneous features and how these features evolve over time. This has to be kept in mind when observing such an animation.

The field lines in our solution is to be regarded as advanced glyphs, which give the direction of the instantaneous vector field at each instant of time and not the direction of the actual path lines. During the animation, these instantaneous curves or glyphs are tracked along the time-dependent vector field. This is done by tracking the center point of the curve along its path line. This means that particle motion is only observed by watching a sequence of images. However, in some cases these two curves can work as approximations for each other, but this depends strongly on the unsteadiness of the flow and the length of the curve. We could of course, just as the UFAC method proposes, vary the length of the field lines according to the unsteadiness of the flow field [24]. When using such an approach, regions of the flow with a high degree of unsteadiness would be very short or even degenerate to a single point. However, since one of the goals with this technique was to give a sparse description of the instantaneous topology of the entire vector field, we have decided to use the same physical length on each field line in the domain.

The advantage with our approach compared to plain use of path lines, is that we are able to clearly convey the instantaneous topology of a time-dependent vector field by for instance visualizing stream lines at each time step. Short path lines can give a similar effect, but, in general, this is not an ideal solution as these curves tend to overlap each other in time, especially when they are relatively long. More importantly, a long path line does not give information about the instantaneous state of the flow. It only gives information about the history of that individual particle. Another advantage with our technique is that it can be used to animate other instantaneous properties such as vorticity lines.

8 Volume Rendering

During the stages of particle advection and the field line generation, all vector data is converted to a series of byte scalar data sets carrying information of all the advected particles and their resulting field lines. This leads to a reduction in storage requirements by a factor of 12, assuming the original vector data was represented as floats. Once these first two stages of the visualization pipeline are finished, the output which is a time series of 3D textures is sent to our rendering framework. Since the rendering stage is decoupled completely from the first two stages, it will ensure a faster rendering compared to algorithms that have to compute texture advection in addition to volume rendering on the graphics card. Also, such texture advection algorithms have to handle much more data including the vector data, all which need to be stored in texture memory. Our algorithm is therefore capable of handling larger volumes interactively than the combined advection and rendering algorithms.

The volumetric data sets are rendered using a standard 3D texture-based direct rendering approach [29], [30]. A stack of view-aligned slicing polygons, serving as proxy geometries, are used to sample the volume and blended together in a back-to-front order to create the final image.

8.1 Multifield Visualization

Since the calculations on the GPU are kept to a minimum during the rendering, we can afford to do some interesting multifield visualizations without loosing interactivity for moderate sized volumes (see Section 9). Our volume rendering framework currently supports four kinds of textures which are luminance (L), alpha (A), mask (M), and custom (“any letter”).

Luminance: This is the standard type of texture. Each value in the data set is used to define both the color and opacity in the final image.

Alpha: The data set values will act as alpha values for the previous luminance texture.

Mask: Multiplies the combined alpha values of all previous textures by the values associated with the mask texture.

Custom: This texture can be used for users who want to write custom built shaders.

Using these four types of textures, we can create a number of different visualizations scenes. To facilitate interactive displays, all supported scenes are written as various shader programs. This could either be as a fragment program or via a high-level shading languages such as Cg [31] and GLSL [32]. We have chosen to write all of the shaders in a high-level shading language which is more human-readable and easier to maintain.

In order to identify the different scenes, we have decided to use a system where each type of texture is identified as a letter (see above). So, if we want to visualize the “particle and field line” (PFL) volume together with velocity magnitude as two separate luminance textures, the shader program used would be identified as LL. A scene where both of these fields are masked by either the velocity magnitude or a third property field would be identified as LLM. The LLM scene can be written in Cg as the following code fragments.

```
float4 L1 = tex3D(voxture1,texCoord1);
float4 L2 = tex3D(voxture2,texCoord2);
float4 M = tex3D(voxture3,texCoord3);
// LUTs
L1 = tex1D(lookup1,L1.r);
L2 = tex1D(lookup2,L2.r);
M = tex1D(lookup3,M.a);
// Blending and masking
float A = L1.a*(1.0 - L2.a) + L2.a;
OUT.color.rgb = (L1.rgb*L1.a + L2.rgb*L2.a)/A;
OUT.color.a = A * M.a; // Mask result by M
return OUT;
```

Fig. 6 shows examples of four different visualization scenes. In Fig. 6a, the PFL texture is visualized as a single luminance texture. In Fig. 6b, the velocity magnitude and the PFL volume are rendered as a luminance/alpha (LA) texture. Fig. 6c shows both the first two scenes blended together, while Fig. 6d shows the previous scene masked by the velocity magnitude calling the final shader program L(LA)M. All textures used in a scene are associated with a separate lookup table (LUT) enabling the user to create a various number of visualization effects.

8.2 Time-Varying Candy Cane Visualization

An alternative way of visualizing vector fields is by using a technique called “candy cane” visualization. The technique was proposed by Helgeland and Andreassen [10], and was further demonstrated in [33]. This, however, is the first time this technique has been used on time-dependent data. The candy cane visualization is done by letting a scalar field define the structure or “body” through opacity and the PFL volume define the color. This two-field visualization technique is an excellent method for visualizing directional information of a three-dimensional vector field inside a region of interest defined by a scalar field. The technique is especially useful when used in conjunction with interactive clip planes, enabling the user to follow field lines from a vector field inside volume data features such as the hurricane shown in Fig. 7. An example of the candy-cane technique is shown in Fig. 7a. Here, the velocity magnitude has been used to define the opacity while the velocity field through the PFL volume has been used to define the color. The candy cane visualization can be done by creating an LA-scene. In Fig. 7d, we have in addition color-encoded the individual field lines according to the velocity magnitude.

A candy-cane animation will show the advection of volume data features “painted” by the advected field lines, leaving the scalar field in focus. This means, that it is the motion of the data features that defines the smoothness of the animation and not the movement of the individual particles or field lines. For the hurricane data set displayed in Fig. 7, sufficiently smooth animation was achieved without increasing temporal coherence (see Section 6).

Similarly, we can create other multifield visualizations with slightly different effects. In Fig. 7b and Fig. 7c, the

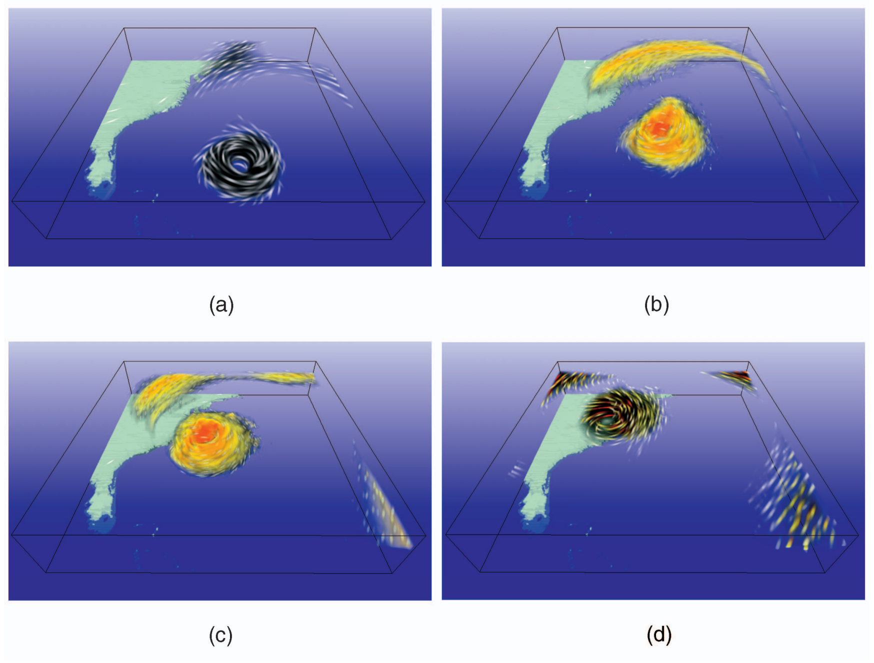

Fig. 7. Illustration of various multifield visualizations using a combination of field lines and dense flow features. Both the flow features and the field lines have been highlighted by a region of interest, in this case, according to the velocity magnitude. (a) Candy-cane visualization of hurricane at $t = 0$. (b) Multifield visualization of hurricane at $t = 16$. (c) Multifield visualization of hurricane at $t = 32$. (d) Candy-cane visualization of hurricane at $t = 48$.

same two fields are visualized independently using two separate color and opacity tables. Here, the region of interest structures are colored by the velocity magnitude directly and not the individual field lines.

8.3 Animation

Once the desired data is selected and an appropriate visualization scene is created, our rendering framework handles two types of navigations through the time-varying data set. The data set can be explored by either dragging a time slider or by using the animation utility. The time slider is very useful for investigating the data at different time steps. Once a new time step is selected, the visualization scene is automatically updated using the same lookup tables. This time slider feature also simplifies the process of finding appropriate color and opacity tables that is well-suited for the whole time-series. Finding good transfer functions is often a tedious process sometimes involving clipping of data value ranges.

Using the animation utility allows a more continuous visualization of the time-dependent data. Here, the user can choose the order of the data sets to be loaded as well as the step size. We believe such a rendering framework allows a more complete investigation of time-varying data sets.

Whenever multiple data sets are loaded, they are assumed to have time-ranges that cover the same physical time domain, even if they have different number of time steps (see Section 6). To ensure correct loading of data, a global time is calculated and used to determine the local time on each individual property field.

8.4 Interactive Analysis

In addition to manipulating the time slider, our rendering framework supports an additional set of tools to facilitate the analysis of a three-dimensional time-dependent flow field. This includes manipulation of transfer functions, clip planes, data-subset selections, and other user functions at interactive rates. These are all tools that can be used to diminish the occlusion effects, by for instance creating transparent visualizations and reducing the complexity of the scene by focusing on a region of interest. When used in conjunction with the sparse and global representation of the volumetric vector field, which is also a technique to reduce occlusion, these tools form a strong basis for overcoming the perceptual problems related to volume visualization.

In order to capture very small features in an even more complex field than the vector fields presented here, it might be necessary to use a higher field line density than the one used in Fig. 6. In such a visualization, due to higher occlusion, it could be more difficult to see deep into the domain. However, even for such fields we believe that the additional set of tools can be used to help overcoming the problem of occlusion and cluttering. A very dense vector field representation is in any case preferable compared to a completely dense representation.

9 Performance

Our implementation is based on C++ and OpenGL and was tested on a Pentium 3.40GHz PC (2048MB RAM) with a nVidia GeforceFX 6800 GT graphics card (256MB). Table 1 shows the total computational time (measured in wall clock time) for the three stages of the time-varying vector field visualization algorithm on three different data sets: heli (30 time steps of a synthetic vector field), a Weather Research and Forecasting (WRF) model (80 time steps), and the Hurricane Isabel (48 time steps) data sets. The advection and field line generation were computed using a field line

TABLE 1
Performance Measurements for the Time-Varying Vector Field Visualization Algorithm

Data Set	Heli ($128 \times 128 \times 128$)	Heli ($256 \times 256 \times 256$)	Heli ($512 \times 512 \times 128$)	WRF ($127 \times 127 \times 29$)	Isabel ($500 \times 500 \times 64$)
No Sub-sampling:					
Particle Advection:					
Initial Distribution. (sec.)	18.8	176.5	374.6	4.0	189.9
Advection (sec.)	17.8	226.9	533.0	7.1	366.6
Field Line Gen.:					
Direct Approach (sec.)	5.5	42.9	84.9	4.0	79.2
With sub-sampling:					
Particle Advection:					
Initial Distribution (sec.)	3.28	31.0	65.6	0.7	30.6
Advection (sec.)	10.2	170.1	407.7	5.1	331.9
Field Line Gen.:					
Direct Approach (sec.)	5.1	39.0	77.3	3.6	65.7
Convolution	26.3	226.3	575.6	15.7	302.8
Rendering:					
L (frames/sec.)	54.9	20.0	12.5	85.5	11.0
LL (frames/sec.)	21.3	6.8	5.4	40.3	5.2
LLM (frames/sec.)	18.0	5.3	3.9	29.8	3.6
LLAM (frames/sec.)	13.7	4.1	2.8	22.6	2.7

length of 18 times the size of the grid cells for all data sets. The inserting separating distance, d_i, was set to four times the grid cell size and the removing separating distance, d_r, was set to one half of the inserting separating distance. The table shows the timings for the advection and field line generation both with and without subsampling. The subsample was taken uniformly using a step size of two in each direction. This resulted in an initial distribution of approximately 10 percent fewer points. A significant part of the particle advection algorithm from Section 4 is spent on the initial seeding distribution (Section 4.1). As a result, both timings from the initial seeding step and the rest of the particle advection stages combined are shown in the table. Once the initial seed points have been distributed into the physical domain, the algorithms described in Fig. 2 and Fig. 3 are only used for maintaining the particle density. This requires significantly less computational efforts compared to the initial distribution. The optional convolution step occuring after the direct field line generation also takes a considerable amount of time, as it has to be computed once for each of the output volume textures. Fortunately, the convolution operation is very easy to parallelize and, for practical purposes, we have an OpenMP version which strongly reduces the computational time.

Once the preprocess stages of the visualization pipeline are completed, the output data is sent to the rendering pipeline which performs texture-based volume rendering. The rendering speed is only dependent of the texture size and the complexity of the shader programs. In Table 1, the rendering performance is shown for four different visualization scenes written as four different fragment programs. The table shows the average frames per second for the whole animation including continuous transfer of 3D textures from main memory to the GPU in addition to volume rendering. All data used to create the visualization scenes were downloaded into main memory before the performance measurements where taken. For all renderings, the volume has been sampled using one slice polygon per voxel and the viewport size is set to 600^2 pixels. From the table, it is clear that we are able to achieve interactive rendering of data sets of resolution $512 \times 512 \times 128$, even when rendering multiple textures simultaneously. Our rendering framework is also capable of rendering non-power of two data sets, but at a cost of rendering performance due to internal data bricking. To utilize multiple rendering pipelines to realize the full potential of modern graphics hardware, we have developed a multipipe and multithread version of our rendering framework using the SGI OpenGL Multipipe SDK (MPK) API [34]. On average, we got an increase in rendering performance by a factor of 1.81 when using the multipipe version on two ATI FireGL X2 (256MB) graphics cards.

10 Conclusion and Future Work

We have presented an interactive texture-based method for visualizing unsteady vector fields, capable of generating a global but sparse representation of the flow. In our solution, each time step in the animation gives a representation of the instantaneous vector field in form of evenly-spaced field lines. This gives still images that are very easy to interpret and are very optimal for diminishing perceptual issues such as occlusion and cluttering. Depth relations among field lines are obtained using limb darkening. Furthermore, orientational as well as directional information can be encoded into the final image using the direct field line generation approach.

By decoupling the rendering stage of the visualization pipeline from the stages of particle advection and field line generation, we ensure a minimum of computational efforts in the rendering stage. Such a solution therefore increases the rendering performance compared to the solutions that have to compute texture advection in addition to volume rendering on the graphics card. Our solution does not have to trade accuracy and quality of the visualization for speed, such as most GPU-based advection methods, as the particle advection and field line generation are handled as a preprocess step.

Since the GPU calculations are kept to a minimum, we can afford doing some interesting multifield visualizations without loosing interactivity for moderate sized volumes. In addition to vector data analysis, we are able to do a more complete investigation of flow data. By choosing various visualization scenes, the users can explore both scalar and vector data simultaneously. In addition, each frame in our time-varying vector field animation algorithm reveals the true instantaneous direction of the flow and not just an approximation of it in form of short path lines. Another advantage with our approach is that we are able to visualize any instantaneous vector field including the velocity field and the vorticity field.

To utilize multiple rendering pipelines, we have developed a multipipe version of our rendering framework which speed up the pixel fill rate. We will continue to explore further benefits of having multiple graphics cards. Various existing and upcoming volume rendering techniques can also be incorporated into our renderer. Another possible extension is to develop hardware-accelerated versions of the preprocess stages of the visualization pipeline.

Acknowledgments

The authors would like to thank Drs. Øyvind Andreassen, Atle Ommundsen, and Bjørn Anders Pettersson Reif at FFI (Norwegian Defence Research Establishment) and Dr. Xing Cai (Simula Research Lab and University of Oslo) for their valuable contributions and feedback. Dr. Greg Hakim and Mr. Rolv Erlend Bredesen at the University of Washington (Department of Atmospheric Sciences) and the US National Center for Atmospheric Research (NCAR) are also acknowledged for providing the WRF and Isabel data sets. Thanks also go to the anonymous reviewers for their comments and suggestions.

References

[1] J.P.M. Hultquist, "Interactive Numerical Flow Visualization Using Stream Surfaces," *Computing Systems in Eng.*, vol. 1, nos. 2-4, pp. 349-353, 1990.

[2] N. Max, B. Becker, and R. Crawfis, "Flow Volumes for Interactive Vector Field Visualization," *Proc. IEEE Visualization Conf. '93,* pp. 19-24, Oct. 1993.

[3] J.J.V. Wijk, "Spot Noise-Texture Synthesis for Data Visualization," *Computer Graphics Proc.*, ann. conf. series, vol. 25, pp. 309-318, July-Aug. 1991.

[4] B. Cabral and C. Leedom, "Imaging Vector Fields Using Line Integral Convolution," *Computer Graphics Proc.*, ann. conf. series, vol. 27, pp. 263-270, July 1993.

[5] U. Diewald, T. Preusser, and M. Rumpf, "Anisotropic Diffusion in VectorField Visualization on euclidean Domains and Surfaces," *IEEE Trans. Visualization and Computer Graphics*, vol. 6, no. 2, pp. 139-149, Apr.-June 2000.

[6] B. Jobard, G. Erlebacher, and M.Y. Hussaini, "Hardware-Accelerated Texture Advection for Unsteady Flow Visualization," *Proc. IEEE Visualization Conf. '00,* pp. 155-162, Oct. 2000.

[7] J.J. Van Wijk, "Image Based Flow Visualization," *Proc. 29th Ann. Conf. Computer Graphics and Interactive Techniques*, pp. 745-754, 2002.

[8] U. Bordoloi and H.-W. Shen, "Hardware Accelerated Interactive Vector Field Visualization: A Level of Detail Approach," *Proc. Eurographics 2002, Computer Graphics Forum*, vol. 21, no. 3, pp. 605-614, Sept. 2002.

[9] V. Interrante and C. Grosch, "Visualizing 3D Flow," *IEEE Computer Graphics and Applications,* vol. 18, no. 4, pp. 49-53, July-Aug. 1998.

[10] A. Helgeland and Ø. Andreassen, "Visualization of Vector Fields Using Seed LIC and Volume Rendering," *IEEE Trans. Visualization and Computer Graphics*, vol. 10, no. 6, pp. 673-682, Nov.-Dec. 2004.

[11] G. Li, U. Bordoloi, and H. Shen, "Chameleon: An Interactive Texture-Based Rendering Framework for Visualizing Three-Dimensional Vector Fields," *Proc. IEEE Visualization Conf. '03,* Oct. 2003.

[12] L. Forssell, "Visualizing Flow over Curvelinear Grid Surfaces Using Line Integral Convolution," *Proc. IEEE Visualization Conf. '94,* pp. 240-247, Oct. 1994.

[13] H.-W. Shen and D. Kao, "UFLIC: A Line Integral Convolution Algorithm for Visualizing Unsteady Flows," *Proc. IEEE Visualization Conf. '97,* pp. 317-323, 1997.

[14] N. Max and B. Becker, "Flow Visualization Using Moving Textures," *Proc. ICASW/LaRC Symp. Visualizing Time-Varying Data,* 1995.

[15] B. Jobard, G. Erlebacher, and M.Y. Hussaini, "Lagrangian-Eulerian Advection of Noise and Dye Textures of Unsteady Flow Visualization," *IEEE Trans. Visualization and Computer Graphics,* vol. 8, no. 3, pp. 211-222, July/Sept. 2002.

[16] D. Weiskopf, G. Erlebacher, M. Hopf, and T. Ertl, "Hardware-Accelerated Lagrangian-Eulerian Texture Advection for 2D Flow Visualization," *Proc. Vision, Modeling, and Visualization Conf.,* Nov. 2002.

[17] A. Telea and J.J. Van Wijk, "3D IBFV: Hardware Accelerated 3D Flow Visualization," *Proc. IEEE Visualization Conf. '03,* pp. 233-240, Oct. 2003.

[18] D. Weiskopf, M. Hopf, and T. Ertl, "Hardware-Accelerated Visualization of Time-Varying 2D and 3D Vector Fields by Texture Advection via Programmable Per-Pixel Operations," *Proc. Vision, Modeling, and Visualization Conf.,* Nov. 2001.

[19] C. Rezk-Salama, P. Hastreiter, C. Teitzel, and T. Ertl, "Interactive Exploration of Volume Line Integral Convolution Based on 3D-Texture Mapping," *Proc. IEEE Visualization Conf. '99,* pp. 233-240, 1999.

[20] D. Weiskopf, T. Schafhitzel, and T. Ertl, "Real-Time Advection and Volumetric Illumination for the Visualization of 3D Unsteady Flow," *Proc. EG/IEEE TCVG Symp. Visualization Eurovis '05,* 2005.

[21] S. Guthe, S. Gumhold, and W. Straßer, "Interactive Visualization of Volumetric Vector Fields Using Texture Based Particles," *Proc. Int'l Conf. Central Europe Computer Graphics and Visualization (WSCG),* 2002.

[22] H. Shen, G. Li, and U. Bordoloi, "Interactive Visualization of Three-Dimensional Vector Fields with Flexible Appearance Control," *IEEE Trans. Visualization and Computer Graphics,* vol. 10, no. 4, pp. 434-445, July/Sept. 2004.

[23] A. Sundquist, "Dynamical Line Integral Convolution for Visualizing Streamline Evolution," *IEEE Trans. Visualization and Computer Graphics,* vol. 9, no. 3, pp. 273-282, July/Sept. 2003.

[24] D. Weiskopf, G. Erlebacher, and T. Ertl, "A Texture-Based Framework for Spacetime-Coherent Visualization of Time-Dependent Vector Fields," *Proc. IEEE Visualization Conf. '03,* pp. 107-114, 2003.

[25] D. Weiskopf, F. Schramm, G. Erlebacher, and T. Ertl, "Particle and Texture Based Spatiotemporal Visualization of Time-Dependent Vector Fields," *Proc. IEEE Visualization Conf. '05,* pp. 639-646, 2005.

[26] B. Jobard and W. Lefer, "Creating Evenly-Spaced Streamlines of Arbitrary Density," *Visualization in Scientific Computing '97. Proc. Eighth Eurographics Workshop,* pp. 43-56, 1997.

[27] D. Stalling and H.-C. Hege, "Fast and Resolution Independent Line Integral Convolution," *Proc. Computer Graphics Conf.,* ann. conf. series, pp. 249-256, Aug. 1995.

[28] R. Wegenkittl, E. Gröller, and W. Purgathofer, "Animating Flow Fields: Rendering of Oriented Line Integral Convolution," *Proc. Computer Animation Conf. '97,* pp. 15-21, June 1997.

[29] B. Cabral, N. Cam, and J. Foran, "Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware," *Proc. 1994 Symp. Volume Visualization,* pp. 91-98, Oct. 1994.

[30] T.J. Cullip and U. Neumann, "Accelerated Volume Reconstruction with 3D Texture Mapping Hardware," Technical Report TR93-027, Dept. of Computer Science, Univ. of North Carolina, May 1994.

[31] R. Fernando and M.J. Kilgard, *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics.* Addison-Wesley Longman Publishing Co., Inc., 2003.

[32] R.J. Rost, *OpenGL(R) Shading Language.* Addison Wesley Longman Publishing Co., Inc., 2004.

[33] A. Helgeland, Ø. Andreassen, A. Ommundsen, B. Pettersson-Reif, J. Werne, and T. Gaarder, "Visualization of the Energy-Containing Turbulent Scales," *Proc. IEEE/SIGGRAPH Symp. Volume Visualization and Graphics,* pp. 103-109, Oct. 2004.

[34] "OpenGL Multipipe SDK," http://www.sgi.com/products/software/multipipe/sdk/, 2006.

Anders Helgeland received the BS and MS degrees in computer science from the University of Oslo, Norway, in 2000 and 2002. He is a PhD candidate in computer science at the University Graduate Center, Norway. His current research interests include flow visualization and volume visualization. He was a recipient of the best poster award at the Proceedings of the IEEE Visualization Conference 2002. He is a member of the IEEE and the IEEE Computer Society.

Thomas Elboth received the bachelors degree in computer science in 2003. In 2005, he received the masters degree in applied math. Both degrees were obtained from the University of Oslo, Norway. Currently, he is employed as a R&D geophysicist at Fugro Geoteam, where he works on seismic imaging. His research interests are in areas of scientific visualization and numerical methods.