Accelerated Unsteady Flow Line Integral Convolution

Zhanping Liu, *Member*, *IEEE Computer Society*, and Robert J. Moorhead II, *Senior Member*, *IEEE*

Abstract—Unsteady flow line integral convolution (UFLIC) is a texture synthesis technique for visualizing unsteady flows with high temporal-spatial coherence. Unfortunately, UFLIC requires considerable time to generate each frame due to the huge amount of pathline integration that is computed for particle value scattering. This paper presents Accelerated UFLIC (AUFLIC) for near interactive (1 frame/second) visualization with 160,000 particles per frame. AUFLIC reuses pathlines in the value scattering process to reduce computationally expensive pathline integration. A flow-driven seeding strategy is employed to distribute seeds such that only a few of them need pathline integration while most seeds are placed along the pathlines advected at earlier times by other seeds upstream and, therefore, the known pathlines can be reused for fast value scattering. To maintain a dense scattering coverage to convey high temporal-spatial coherence while keeping the expense of pathline integration low, a dynamic seeding controller is designed to decide whether to advect, copy, or reuse a pathline. At a negligible memory cost, AUFLIC is 9 times faster than UFLIC with comparable image quality.

Index Terms—Flow visualization, vector field visualization, line integral convolution, unsteady flows, UFLIC, texture synthesis, image convolution, acceleration.

1 Introduction

Unsteady flow visualization remains a challenging research topic and plays an important role in computational fluid dynamics. Arrow plots, streamlines, and stream surfaces are straightforward approaches for steady flow visualization. Commodity graphics cards can be leveraged to accelerate rendering of primitives for real-time visualization. However, inappropriate seed placement may either produce cluttered images or display the field only at a local, discrete, and coarse level. To address this problem, van Wijk proposed a texture synthesis technique, called spot noise [1], to visualize flow data by distributing elliptic texture splats within the field and warping the spots in the flow direction. Later Cabral and Leedom [2] presented line integral convolution (LIC) to compute each output pixel value by convolving a white noise texture along the windowed streamline symmetrically advected in both directions from the pixel location. LIC synthesizes an image that provides a global dense representation of the flow, analogous to the resulting pattern of wind-blown sand. Since then, there have been many optimizations of and extensions to LIC such as fast and resolution independent LIC [3], parallel LIC [4], LIC on curvilinear grids [5], LIC on triangulated surfaces [6], magnitude LIC based on multi-frequency noise [7], oriented LIC [8], enhanced LIC with flow feature detection [9], LIC with dye advection [10], volume LIC [11], [12], and HyperLIC for tensor field visualization [13]. Heidrich et al. [14] incorporated indirect pixel texture addressing and additive/subtractive texture blending to accelerate streamline integration and texture convolution in LIC. Preuber and Rumpf [15] adopted another texture-based method, anisotropic nonlinear diffusion, to low-pass filter a noise texture along streamlines, but enhance edges in the orthogonal flow direction. Later Burkle et al. [16] adapted this approach with texture transport for unsteady flows.

The tremendous advances in computing power and storage capacity enable high-fidelity numerical simulation of unsteady flows. Unsteady flow visualization provides more insight into the flow evolution than can be revealed using instantaneous visualization techniques. Pathlines and streaklines [17] are often used to show particle traces in time-varying flow fields. Unsteady flow volumes [18], the adaptive tetrahedralization of a union of streaklines, are an intuitively more understandable representation of 3D evolving flows. Max and Becker [19] applied texture mapping in either forward mesh warping or backward texture coordinates advection for time-dependent flow visualization. Forssell and Cohen [5] used pathlines as convolution paths in LIC to visualize unsteady flows. Verma et al. [20] presented pseudo-LIC in which pre-synthesized template textures are mapped on sparsely placed pathline ribbons to emulate a dense representation of time-dependent flows. Some methods take advantage of hardware capabilities to achieve high-performance visualization. Jobard et al. [21], [22] designed an efficient rendering pipeline based on indirect pixel texture addressing [14] for fast texture/dye advection and feature extraction. Weiskopf et al. [23] exploited the pixel texture unit to visualize time-varying 3D vector fields.

Van Wijk proposed IBFV [24] in which a sequence of temporally-spatially low-pass filtered noise textures are advected via forward texture mapping on warped meshes in combination with blending of successive frames. This easy-to-implement, efficient, versatile method can emulate a wide range of techniques such as particles, arrow plots, streamlines, timelines, spot noise, LIC, and topological

• *The authors are with the ERC/GeoResources Institute, Mississippi State University, PO Box 9627, Mississippi State, MS 39762-9627. E-mail: {zhanping, rjm}@erc.msstate.edu.*

Manuscript received 25 June 2003; revised 29 Apr. 2004; accepted 12 May 2004; published online 13 Jan. 2005.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-0048-0603.

analysis at high frame rates using basic hardware features. IBFV was then used to visualize flow on 3D curved surfaces and enhance surface shape cuing by means of flow-aligned textures [25]. IBFV was even extended to 3D flows by decomposing the 3D advection to planar and longitudinal advections [26]. Jobard et al. presented LEA [27] to visualize unsteady flow fields also at high frame rates despite its independence of hardware acceleration [28]. To compute each pixel value of a frame, backward integration is used to retrieve, at the last time step, the contributing particle's footprint into which the iteratively advected texture, with noise injected at in-flow areas, is indexed for the contributed texture value. Successive textures are blended to create temporal coherence along pathlines in the flow evolution and short-length directional low-pass filtering is performed to improve spatial coherence along instantaneous streamlines. Visually pleasing effects can be obtained when arbitrarily shaped field domains are addressed using contextual masking. Recently, Laramee et al. [29] presented a novel method based on IBFV and LEA for a direct dense representation of unsteady flows on arbitrary triangular surfaces to address large, unstructured, dynamic meshes by projecting the surface geometry to image space to which backward mesh advection and successive texture blending are applied. Weiskopf et al. proposed UFAC [30], which, based on a generic spacetime-coherent framework, establishes temporal coherence by property advection along pathlines while building spatial correlation by texture convolution along instantaneous streamlines. A well-known hardware-independent algorithm is unsteady flow LIC (UFLIC) proposed earlier by Shen and Kao [31]. UFLIC uses a time-accurate value scattering scheme and a successive texture feed-forward strategy to achieve very high temporal and spatial coherence. At each time step, a value scattering process occurs for which a seed is placed in each pixel. For this seed, a pathline is integrated along which the seed's texture value is scattered to the downstream pixels over several time steps. The values received by each pixel at a time step are accumulated and convolved to synthesize the corresponding frame.

IBFV, LEA, UFAC, and UFLIC are the most competitive methods for visualizing unsteady flow fields, each with its own advantages and disadvantages. Each IBFV frame is the result of a line integral convolution of a sequence of images along pathlines. The exponential decay convolution filter used in IBFV to low-pass filter noise textures is well-suited for introducing temporal coherence in the animation; however, the spatial coherence it constructs in each frame may be insufficient. Flow directions are either noisy or artificially blurred [30] as the texture scale varies. Second, the increasing (unsteady) flow complexity greatly compromises the performance unless the field is highly subsampled to create a warping mesh, as was done in [24], [29] in order to achieve high frame rates. Third, 3D IBFV is limited in the range of velocities it can display, as stated in [26]. Fourth, 3D IBFV handles only time-independent 3D flows since time-varying flows require a continuous update of the velocity texture, which is difficult to achieve. Finally, IBFV depends on hardware capabilities coupled with single-step forward integration to achieve high frame rates. LEA also employs single-step integration, though backward, to access the last frame for advected texture values. It resorts to blending successive textures to represent spatial correlation along a dense set of pathline segments to approximate short streamlines, but the exponentially decreasing temporal filter does not produce sufficient spatial coherence either [30]. Despite the application of LIC to suppress aliasing artifacts created where the noise is advected more than one cell per integration, only shorter kernels can be used since streamlines would otherwise significantly deviate from actual pathlines, causing flashing in the animation and degraded image contrast. Thus, there exists a trade off between the spatial coherence in an image and the temporal coherence in the animation. Flow directions are obscure in low-magnitude areas when the length of the streaks is proportional to the velocity magnitude. UFAC was derived from a generic spacetime-coherent framework, which provides an explicit, direct, and separate control over temporal coherence and spatial coherence to emulate IBFV, LEA, and UFLIC. However, as stated in [30], it still fails to resolve the inconsistency between temporal and spatial patterns since the evolution of streamlines along pathlines might not lead to streamlines of the subsequent time step. Its ad hoc solution to this problem is limited to only an explicit control over the length of the spatial structures based on the flow unsteadiness to retain temporal coherence. In regions where the flow changes rapidly, the correlated segments along streamlines have to be very short and even degenerate to points (particles) to suppress flickering, which inevitably affects spatial coherence. Finally, UFAC is limited to DirectX 9.0 compliant GPUs or OpenGL with fragment support (pixel shader programs). UFLIC possesses the advantage of conveying very high temporal and spatial coherence by scattering fed-forward texture values. Value scattering along a long pathline over several time steps not only correlates a considerable number of intraframe pixels to establish strong spatial coherence, but also correlates sufficient interframe pixels to build tight temporal coherence. Texture feed-forward that takes an output frame, after noise-jittered high-pass filtering, as the input texture for the next frame constructs an even closer correlation between the two consecutive frames to enhance temporal coherence. Flow directions are clearly depicted in individual images for instantaneous flow investigation and the animation is also quite smooth (see http://www.erc.msstate.edu/~zhanping/Research/FlowVis/AUFLIC/index.html). The inconsistency between temporal and spatial patterns in IBFV, LEA, and UFAC is successfully resolved by scattering fed-forward texture values in UFLIC. Also, UFLIC can be easily extended to time-varying 3D flows [32]. The low performance of UFLIC is primarily due to intensive pathline computation that typically needs over 100 steps of integration for each pathline. There is a huge potential to accelerate UFLIC by reducing pathline redundancy and integration steps.

In this paper, we present Accelerated UFLIC (AUFLIC), an enhanced version of our earlier work [33], which increases the performance of UFLIC to near interactive frame rates. AUFLIC seeks to accelerate value scattering by circumventing pathline integration as much as possible while maintaining a dense scattering coverage. A flow-driven seeding strategy is used to place seeds sparsely to integrate fewer pathlines. This allows known pathlines to be reused, with minor corrections, to extract the trajectories of seeds subsequently placed downstream along the pathlines.

A dynamic seeding controller is devised to decide whether a pathline is advected from scratch, copied from that of another seed in the current scattering process, reused from the last scattering process, saved for the next scattering process, or finally deleted. Such an adaptive seed placement driven by the flow pattern effectively exploits the computational redundancy of the dense, texture-based flow visualization technique for an order-of-magnitude acceleration while retaining very high temporal and spatial coherence.

This paper is organized as follows: We first revisit pathline integration, give an overview of the UFLIC method, and discuss the bottleneck. Next, we present our algorithm in detail. Then, we give some results to demonstrate the image quality and the acceleration. We finally conclude this paper with a brief summary and outlook on future work.

2 Background

In this section, we first introduce a fast pathline integration approach, the fourth order Runge-Kutta integrator with adaptive step size and error control, for unsteady flow line advection. We then review the UFLIC method and analyze the latency of the pipeline.

2.1 Fourth Order Runge-Kutta Integration (RK4)

With the vector varying with time and space, the pathline advected by a particle is governed by:

$$\mathrm{d}(\rho(\mathrm{t}))/\mathrm{dt} = \nu(\rho(\mathrm{t}), \mathrm{t}),$$

where $\rho(\mathrm{t})$ is the particle's position at time t and $\nu(\rho(\mathrm{t}), \mathrm{t})$ is the vector at $\rho(\mathrm{t})$ at time t.

The particle's trajectory over time is traced by integrating the equation step by step:

$$\rho(\mathrm{t}+\Delta\mathrm{t}) = \rho(\mathrm{t}) + \int_{\mathrm{t}}^{\mathrm{t}+\Delta\mathrm{t}} \nu(\rho(\mathrm{t}), \mathrm{t})\mathrm{dt}.$$

There are three well-known methods with increasing accuracy for the integration:

$$\begin{aligned}
\text{Euler method} \quad & \rho(\mathrm{t}+\mathrm{s}) = \rho(\mathrm{t}) + \mathrm{s} \times \nu(\rho(\mathrm{t}), \mathrm{t}) \\
\text{Midpoint method} \quad & \rho(\mathrm{t}+\mathrm{s}) = \rho(\mathrm{t}) + \mathrm{s} \times \nu(\rho(\mathrm{t}) + \nu(\rho(\mathrm{t}), \mathrm{t}) \\
& \times \mathrm{s}/2, \mathrm{t} + \mathrm{s}/2)
\end{aligned}$$

$$\begin{aligned}
\text{RK4} \quad \Delta\rho_0 &= \mathrm{s} \times \nu(\rho(\mathrm{t}), \mathrm{t}) \\
\Delta\rho_1 &= \mathrm{s} \times \nu(\rho(\mathrm{t}) + \Delta\rho_0/2, \mathrm{t} + \mathrm{s}/2) \\
\Delta\rho_2 &= \mathrm{s} \times \nu(\rho(\mathrm{t}) + \Delta\rho_1/2, \mathrm{t} + \mathrm{s}/2) \\
\Delta\rho_3 &= \mathrm{s} \times \nu(\rho(\mathrm{t}) + \Delta\rho_2, \mathrm{t} + \mathrm{s}) \\
\rho(\mathrm{t}+\mathrm{s}) &= \rho(\mathrm{t}) + \Delta\rho_0/6 + \Delta\rho_1/3 + \Delta\rho_2/3 + \Delta\rho_3/6,
\end{aligned}$$

where $\mathrm{s} = \Delta\mathrm{t}$ is the integration step size.

Temporal-spatial interpolation is performed intensively during the RK4 integration to evaluate intermediate vectors. As a multistage integrator, the RK4 method for unsteady flow advection actually achieves only second-order accuracy when temporal interpolation is based on the assumption that the flow varies linearly between two time steps [17]. Adaptive step-sizing based on local error control is usually incorporated into the RK4 method to accelerate pathline integration with a user-defined error tolerance. A straightforward way to estimate the local error is to compare the two solutions obtained from the Runge-Kutta with consecutive orders. An alternative for fast error estimation is to use embedded Runge-Kutta formulae. Successive positions returned by the RK4 integrator with adaptive step size and error control (RK4-ASSEC) may be further interpolated using a cubic Hermite polynomial for texture sampling by equal distance [34]. The RK4-ASSEC, coupled with cubic Hermite interpolation, is a faster integrator with more accuracy and flexibility than the Euler method for pathline advection.

2.2 Unsteady Flow Line Integral Convolution

When using LIC to visualize a steady flow field, for each pixel of the output image, the contributing pixels are first located along the bidirectionally advected streamline and then the noise texture values are indexed for convolution. Because the forward and backward streamlines are intrinsically correlated in a steady flow, low-pass filtering can be implemented in either image space or object space. However, LIC cannot be directly applied to an unsteady flow to establish spatial coherence since the forward and backward pathlines may be uncorrelated [31]. Also, the resulting animation fails to maintain temporal coherence due to the different convolution paths along which the filter phase is shifted.

UFLIC [31] is an object-space texture synthesis technique designed to visualize unsteady flow fields. The two important components, the time-accurate value scattering scheme and the successive texture feed-forward strategy, are based on the observation that particles leave their footprints, i.e., deposit their properties, at downstream positions as the flow runs over time. At each time step, a scattering process (SCAP, Fig. 1) occurs for which a seed is released, i.e., discharged, from each pixel as a contributor to scatter the texture value to the downstream pixels along the newly advected pathline in its life span that usually ranges through several time steps. On the other hand, as a receiver, each pixel keeps several stamped buckets in a ring buffer to accumulate deposited values. A frame is obtained by convolving the values that each pixel has received and stored in the bucket stamped with the frame index, creating spatial coherence in the output image. To enhance temporal coherence, the resulting texture is high-pass filtered with noise jittering and then fed forward as the input texture to the next SCAP. Fig. 2 shows the UFLIC pipeline.

In the UFLIC pipeline, over 90 percent of the time consumption stems from the SCAP, in particular, from computationally expensive pathline integration that involves intensive temporal-spatial vector interpolation and, usually, pixel-by-pixel line clamping. It is worth mentioning that pathlines advected in an SCAP are not used in the subsequent SCAPs. In other words, the existing correlation between the SCAPs is neglected instead of being exploited. This situation is illustrated in Fig. 1. In SCAP k, a seed released from pixel center A at time step k advects a pathline in the life span to scatter the value to the downstream pixels. B, C, and D are the points through which the seed passes at time step $k+1$, $k+2$, and $k+3$, respectively. A seed, if released from B, C, and D at time step $k+1$, $k+2$, and $k+3$ in SCAP $k+1$, $k+2$, and $k+3$, respectively, would follow the same trajectory as the pathline advected in SCAP k. B, C, and D could be therefore extracted from the pathline as the seeding positions in SCAP $k+1$, $k+2$, and $k+3$ to avoid pathline integration. However, in UFLIC, seeds are always released from pixel

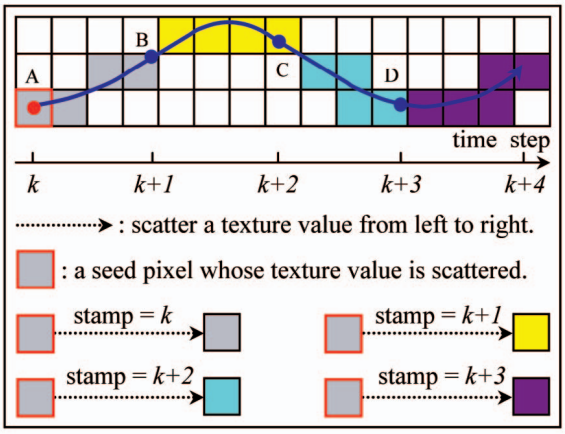


Fig. 1. A seed is released from pixel center A to advect a pathline in SCAP k, which begins at time step k (life span = 4 time steps) and generates frame k. Frame $k-1$ is high-pass filtered with noise jittering and then taken as the input texture for SCAP k (white noise is used for $k=0$).

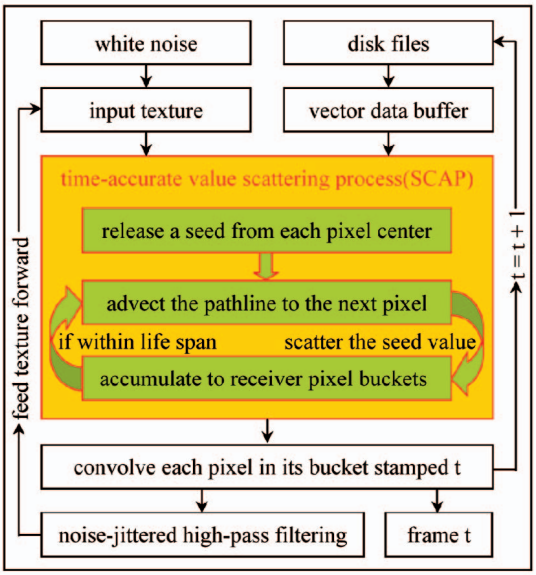


Fig. 2. The UFLIC pipeline.

centers to advect new pathlines, which slows down the value scattering process.

In the following section, we propose an accelerated UFLIC algorithm. It is based on a flow-driven seeding strategy by which fewer seeds are sparsely released in the flow direction to advect only a small number of pathlines along which more seeds are then released to circumvent pathline integration. A dynamic seeding controller is used to decide whether to advect, copy, or reuse a pathline so as

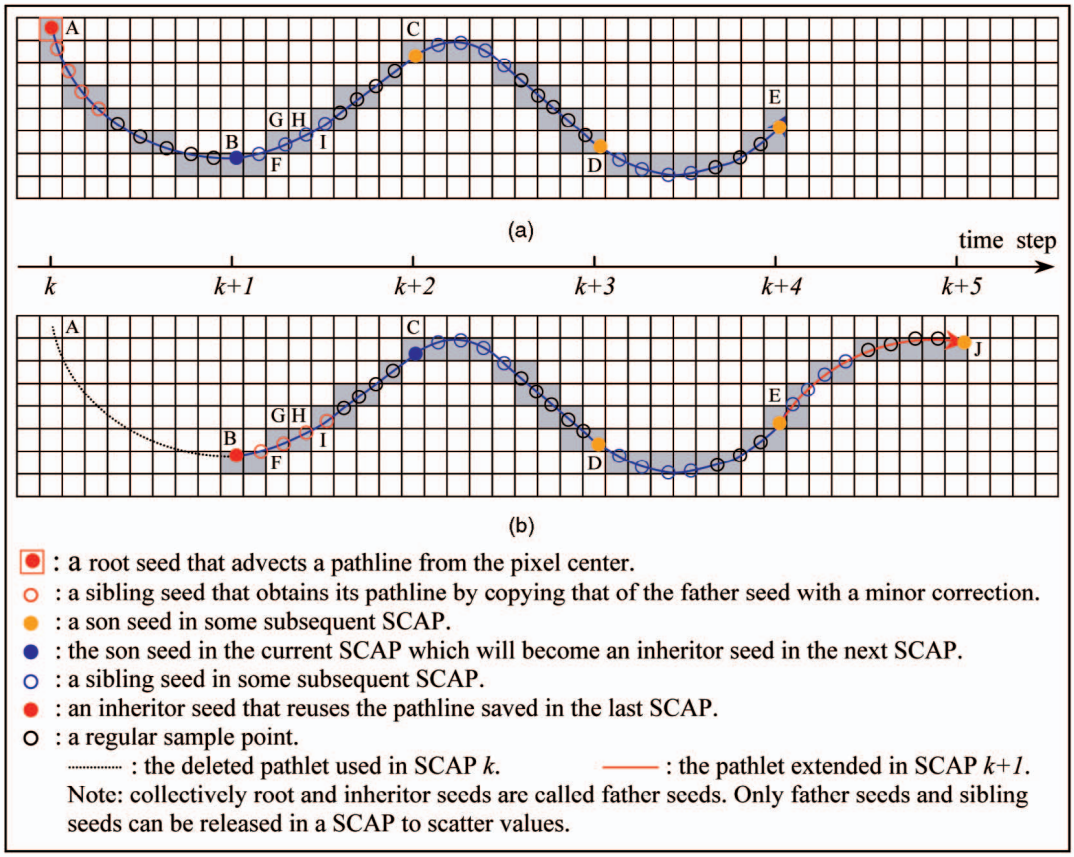


Fig. 3. The flow-driven seeding strategy (life span = 4 time steps). (a) In SCAP k, root seed A is first released from the pixel center to advect a pathline for value scattering. Sibling seeds (hollow red) are then released and the pathline is copied, with minor corrections, to scatter their values. At the end of SCAP k, the first pathlet is deleted. Seeds B (son of A), C (son of B), D, E, other seeds (hollow blue), and regular sample points (hollow black) are saved for SCAP $k+1$. (b) In SCAP $k+1$, B becomes an inheritor seed and C becomes the son seed. The pathline saved in SCAP k is inherited by seed B and is extended with one pathlet forward for complete value scattering.

```
void LocalSemaphoreOperator(FATHERSEED f, PATHLINE l)
{   close the father pixel's CurSema;
    semi-open the father pixel's NxtSema;
    for each of SIBLINGS sibling seeds
    {   if (the sibling pixel's CurSema is closed)  continue;
        release the sibling seed along l;
        close the sibling pixel's CurSema;
    }
    for each of SIBLINGS/2 pixels in the opposite flow direction at f
    {   if (the pixel's CurSema is open) semi-open the CurSema;   }
}
void DynamicSeedingController( )
{   open the NxtSema array;
    for each SCAP
    {   CurSema array = NxtSema array;
        open the NxtSema array;
        for each pathline l saved in the last SCAP
        {   release the inheritor seed i and extend l for a new pathlet;
            LocalSemaphoreOperator(i, l);
            if (the son pixel's NxtSema is not open)  delete l;
            else  save l and close the son pixel's NxtSema;
        }
        for each pixel
        {   if (the pixel's CurSema is not open)  continue;
            release a root seed r from the pixel and advect a pathline l;
            LocalSemaphoreOperator(r, l);
            if (the son pixel's NxtSema is not open)  delete l;
            else  save l and close the son pixel's NxtSema;
        }
    }
}
```

Fig. 4. Pseudocode for the dynamic seeding controller (*SIBLINGS*: the maximum number of sibling seeds).

to scatter enough particles while minimizing pathline integration.

3 AUFLIC

It is mandatory in UFLIC that nearly every pixel receives enough particle values in the value scattering process that pixels can be fully correlated to show temporal-spatial coherence. Insufficient particle scattering implies fewer hits are received and less temporal-spatial correlation is exploited, which may introduce undesirable artifacts such as lack of line smoothness, discontinuities in the flow pattern, missing features, and flashing in the animation. In each SCAP, the UFLIC algorithm advects a new pathline from every pixel center and forcibly terminates the pathline when the life span expires, even if it has not run outside the field or encountered any critical points. If a flexible seeding rule is used, fewer pathlines need to be advected in an SCAP and they may be copied or later reused to obtain traces for other particles to achieve a dense scattering coverage. We adopt such a flexible seeding rule as follows:

- Spatial flexibility: A seed may not necessarily be released exactly from a pixel center as long as the seed is within the pixel.
- Temporal flexibility: A seed may not necessarily be released exactly at an integer time step; instead, it may be released at a fractional time shortly after the SCAP begins.

In this section, we present an accelerated UFLIC algorithm, AUFLIC, that uses a flow-driven seeding strategy to release seeds along known pathlines so that only a few of them need to actually advect pathlines; the rest can simply extract their pathlines using pathline copying and pathline reuse. Pathline copying is an intra-SCAP operation by which a seed's trace in the life span is used, with only minor corrections, for those of other seeds successively released at several positions downstream along the known trace at fractional times shortly after the SCAP begins, as long as they are released at the same time as the initial seed travels through their seeding positions. Each of these seeds travels through a different-length part of the same curve during the first time step of the SCAP, but they synchronously run through the same trace over the remaining time steps. Pathline reuse is an inter-SCAP operation by which the position a pathline passes through within a fractional time into the second time step of the previous SCAP is used to release a new seed at exactly the same global time, but in the first time step of the current SCAP. This seed's trace is obtained by reusing the latter part of the known pathline from the previous SCAP, appended with integration over an additional time step. Pathline copying applies whether a pathline is obtained by reuse or brute-force integration. To fulfill the seeding strategy for a dense scattering coverage at a low computational cost,

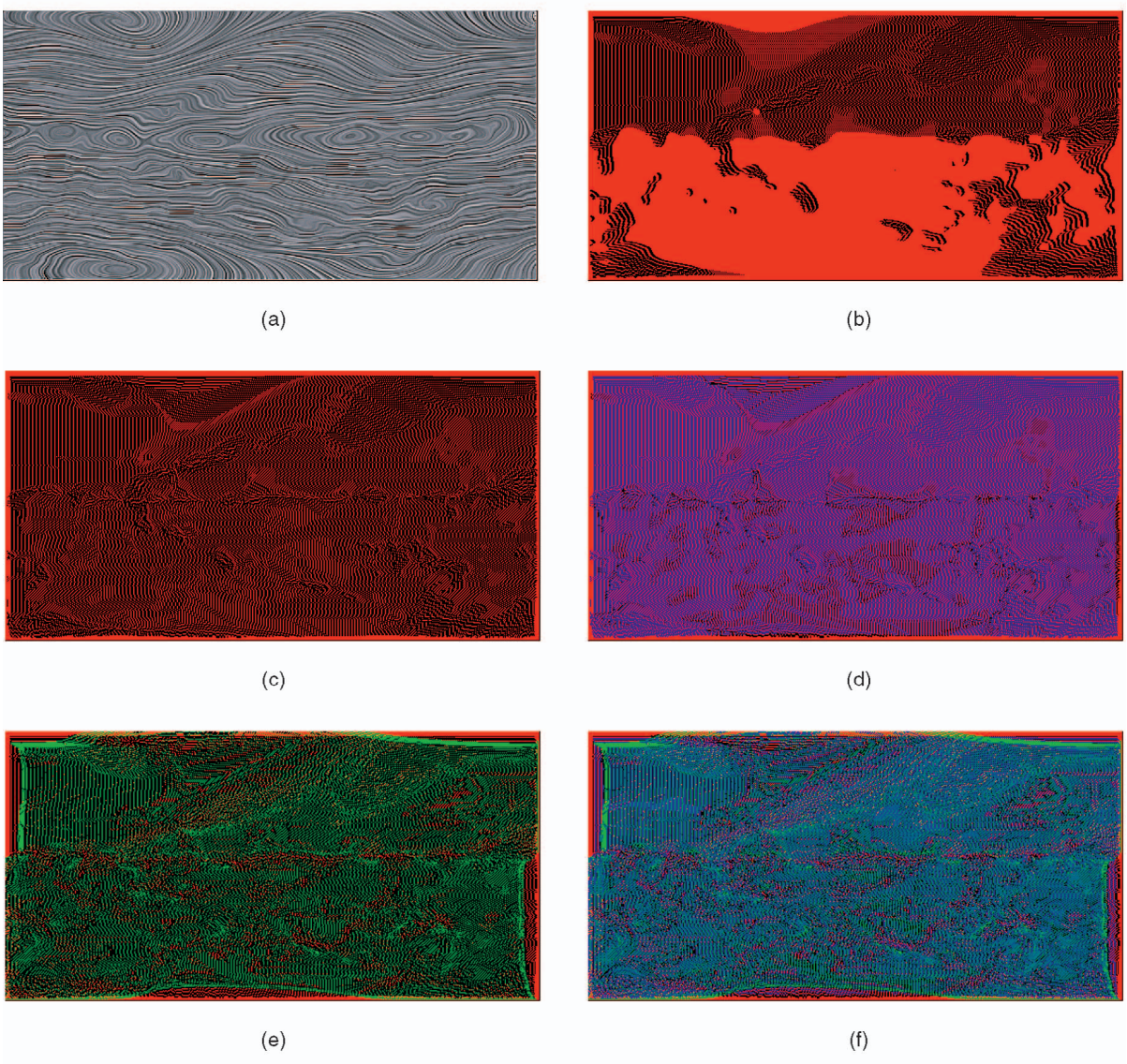
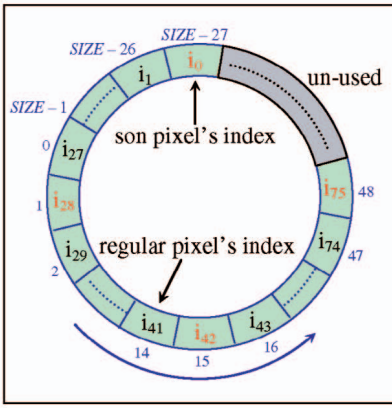


Fig. 5. Seed distribution maps in SCAP 0 and 1 (red, green, and blue pixels for root, inheritor, and sibling seeds, respectively, SIBLINGS = 6). (a) The LIC image of a flow field at time step 0. (b) The seed distribution in SCAP 0 (sibling seeds not displayed) without CurSema semi-opening in the opposite flow direction. The bottom-half flow runs from right to left, which is against the regular order of releasing root seeds. No pixels are reserved for releasing downstream sibling seeds. Note how densely the root seeds are placed. (c) The seed distribution in SCAP 0 (sibling seeds not displayed) with CurSema semi-opening in the opposite flow direction. Note how evenly and sparsely the root seeds are placed. (d) The seed distribution in SCAP 0 (sibling seeds displayed). (e) The seed distribution in SCAP 1 (sibling seeds not displayed). (f) The seed distribution in SCAP 1 (sibling seeds displayed).



Header Information	
pSonPixelPointer[0] = *SIZE* – 27 pSonPixelPointer[1] = 1 pSonPixelPointer[2] = 15 pSonPixelPointer[3] = 48	to retrieve son pixels' indices and demarcate different pathlets in the pixel ring-buffer
pSonSeedCoord[0] = (25.3, 72.8) ..... pSonSeedCoord[3] = (82.5, 36.9)	to interpolate texture, vector for advection & semaphore operator
pSonSeedTime[0] = 6.0235 ..... pSonSeedTime[3] = 9.1308	to advect new pathlets

Fig. 6. Pixel ring-buffer and the header information (life span = 4 time steps).

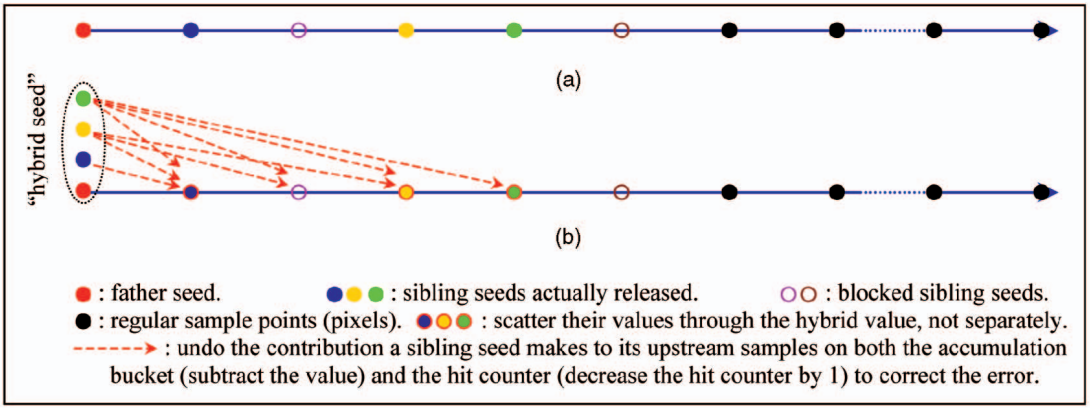


Fig. 7. Using hybrid value scattering to simplify sibling seed value scattering in pathline copying. (a) The father seed, sibling seeds, and regular sample points on a pathline (flattened). (b) Scatter only the father seed using the hybrid value (but increase the hit counter by four each time) in combination with minor corrections to simplify sibling seed value scattering in pathline copying.

AUFLIC uses a dynamic seeding controller to choose for each seed between pathline advection, copying, and reuse.

3.1 Flow-Driven Seeding Strategy

Given the spatial and temporal seeding flexibilities, a pathline can be evenly interpolated using a cubic Hermite polynomial to generate sample points, some of which are chosen as potential seeding positions for several SCAPs. We delay the discussion of whether a seed is actually released from a potential seeding position until Section 3.2. Those not chosen as potential seeding positions are called *regular* sample points. We define seed types (Fig. 3) along an evolving pathline in the SCAPs based on genealogical concepts—birth, propagation, and death. Just as a person's title depends on both his level in the family structure and his role in his generation, a seed's name depends on both its release time relative to the current SCAP and the way the pathline is obtained. In SCAP k (Fig. 3a) that begins at time step k, particle A, called a *father seed*, is the first seed released along a pathline in the current SCAP. Seed A is specifically called a *root seed* because it advects the pathline from scratch. The hollow red particles are *sibling seeds* of the father seed that are released at some fractional times shortly after the SCAP begins (at time step k), from the same positions through which the father seed travels at these times. Except for the difference in release time and, hence, in path length during the first time step of the SCAP, the sibling seeds and the father seed move along the same trace synchronously. Given the pathline of the father seed, those of the sibling seeds are easily extracted by pathline copying with minor corrections. Particles B, C, D, and E are *son seeds*. They will be first released along the pathline in the subsequent SCAPs. Specifically, seed B is the son seed of A and will become the father seed in the next SCAP. The father seed and the sibling seeds, once released, scatter their texture values to the succeeding pixels hit by the pathline before convolution is performed to synthesize frame k. With the first *pathlet* (i.e., the portion of a pathline advected during one time step) cut off, the pathline is saved for use in the next SCAP.

In SCAP $k+1$ (Fig. 3b) beginning at time step $k+1$, seed B, a son seed in SCAP k, becomes a father seed. It is specifically called an *inheritor seed* since it inherits a pathline from SCAP k. It does not need to advect a pathline, but, instead, can simply reuse the partial pathline saved in SCAP k since the two SCAPs overlap from time step $k+1$ to $k+4$. In order to scatter the seed value over the whole life span, the pathline is integrated through another time step to generate an additional pathlet. Seeds F, G, H, and I are released for value scattering along the pathlines that are obtained by copying and slightly truncating the pathline of seed B, their older sibling. Seed C is the elected son seed of seed B and so forth. Only father seeds and sibling seeds can be released in an SCAP. Collectively, root and inheritor seeds are called father seeds. The *father pixel* (of a pathline), for short, refers to the pixel hit by the father seed. Similar pixel names refer to the pixels hit by other kinds of seeds.

As described above, such a *flow-driven seeding strategy* makes it possible to scatter more particles using less pathline advection. To reserve positions for sibling seeds, the flow-driven seeding strategy subsamples the field in the flow direction to release sparse father seeds. However, father seeds are densely released in the orthogonal flow direction to deal with flow divergence. The advantage of the flow-driven seeding strategy over random seeding schemes is that it can be easily implemented using a dynamic seeding controller (Section 3.2) to adaptively choose between pathline advection and pathline reuse for better overall performance. It is difficult for a random seeding scheme to make use of inter-SCAP correlation since it does not take flow structures into account.

The spatial flexibility introduces an arbitrary fractional offset of a seed within the pixel, which may help to reduce artifacts. The texture value is obtained either by bilinear interpolation or by nearest-neighbor indexing. The temporal fractional offset should be small enough that not too many sibling seeds are chosen since a larger temporal offset means a shorter life span. To reuse a pathline in as many SCAPs as possible, it is advected forward until it runs outside the field, encounters a critical point, or is killed by the dynamic seeding controller.

3.2 Dynamic Seeding Controller

Now, we present a dynamic seeding controller to implement the flow-driven seeding strategy, i.e., to achieve a dense scattering (or seeding) coverage with a small number of root seeds and a significant number of inheritor and sibling seeds. The controller governs the seed distribution

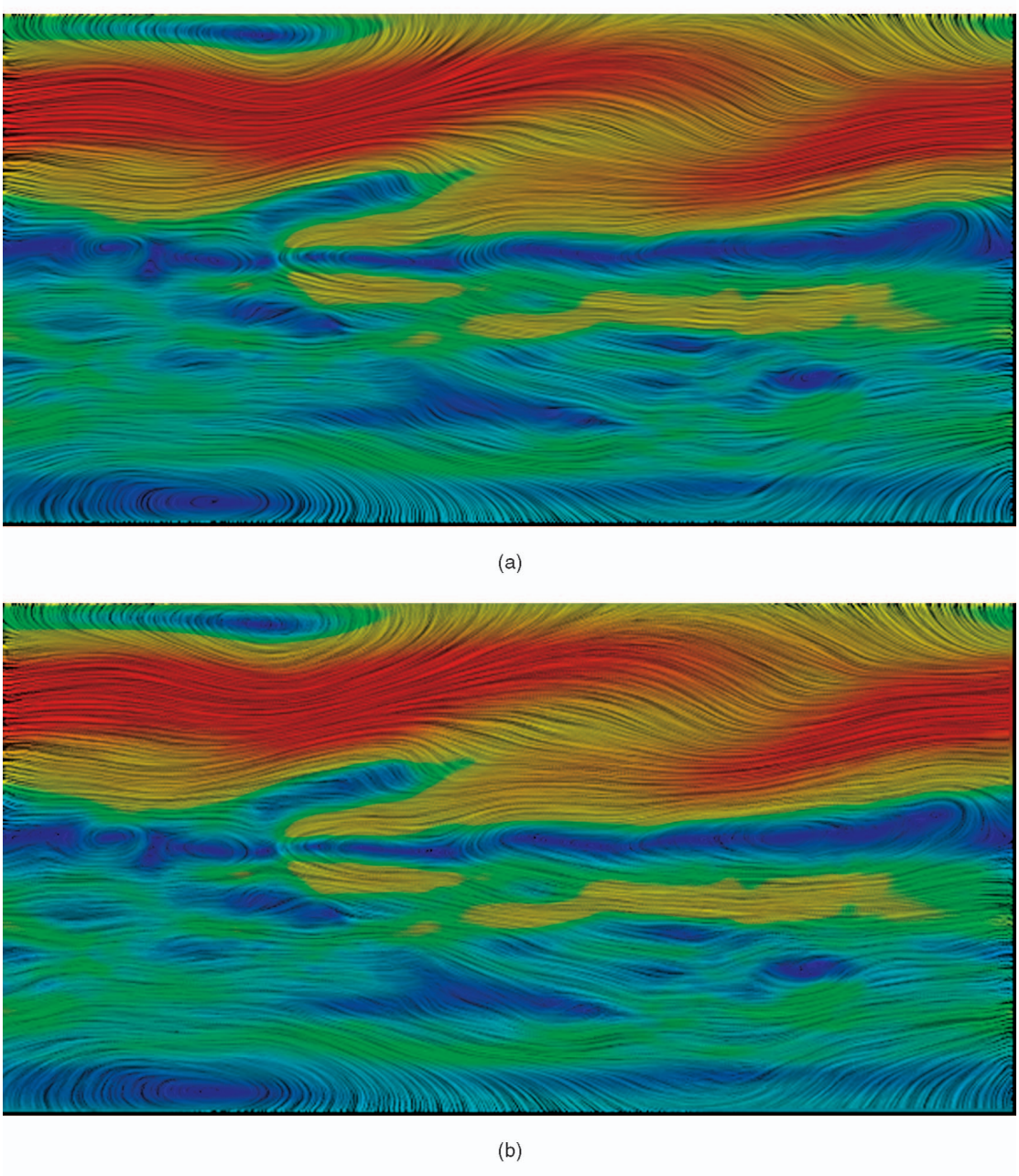


Fig. 8. Two images generated by (a) UFLIC and (b) AUFLIC from the same weather data (576×291 data points).

in an SCAP to decide whether a potential seed is actually released or which seed is actually released from each pixel when many inheritor seeds, many sibling seeds, and a root seed are potentially released from the same pixel in an SCAP. There is a trade off between pathline reuse and advection: The more pathlines reused, the fewer pathlines advected in the current SCAP; the fewer pathlines reused, the more pathlines advected in the next SCAP. This situation implies the computational cost could severely fluctuate over SCAPs. To obtain a nearly constant frame rate, such a controller seeks to balance pathline reuse and advection in each SCAP.

We assign two semaphores, *CurSema* and *NxtSema*, to each pixel to ensure no more than one seed is released from it in an SCAP. Associated with a semaphore are some states and operations:

- *Open state* (0). Either a father seed or a sibling seed can be released from the pixel.
- *Semi-open state* (1). Only a sibling seed can be released from the pixel.
- *Closed state* (2). No more seeds can be released from the pixel.
- *Open operation*. Make a semaphore open.
- *Semi-open operation*. Make a semaphore semi-open.
- *Close operation*. Make a semaphore closed.

A CurSema indicates the pixel status for the *current* SCAP and determines what seed, if any, may be released from the pixel. A NxtSema indicates the pixel status for the *next* SCAP and decides whether a pathline whose son seed falls within the pixel is saved or deleted. When an SCAP begins, the CurSema array is first refreshed with the NxtSema array and

the latter is opened. Then, the two semaphore arrays are dynamically updated as the SCAP runs.

Once a seed is released from a pixel, the CurSema is closed to block further seed release from the pixel in the SCAP. An SCAP begins with releasing inheritor seeds. Then, root seeds are released from the pixels still with open CurSemas in a regular order, i.e., from top to bottom and left to right. Some sibling seeds are released each time a father seed is released. To update semaphores, we define a local operator for each father seed. Given a father seed in a pixel, independent of whether the pathline is advected or reused, we close the CurSema and semi-open the NxtSema so that father seeds and sibling seeds are released from the pixel in alternate SCAPs. In the flow direction, sibling seeds are released from some pixels and the CurSemas are closed. In the opposite flow direction, some pixel CurSemas are semi-opened, if still open, to reserve pixels for sibling seeds in order to deal with a bottom-to-top or right-to-left flow for which the advantage of the flow-driven seeding strategy might otherwise degrade as most sibling seeds would be blocked due to falling within the pixels that are already hit by root seeds in a regular seeding order (Fig. 5b). Along each border of the field, a small margin is usually left without semaphore semi-opening to enable dense seed release from in-flow borders. The pathline is deleted unless the son pixel's NxtSema is open; otherwise, it is saved and the NxtSema is closed.

Fig. 4 shows the pseudocode for the dynamic seeding controller using the local semaphore operator to implement the flow-driven seeding strategy. Root, inheritor, and sibling seeds are systematically distributed for optimal performance. A significant amount of inheritor seeds are released to reuse the pathlines saved in the last SCAP, while only a small number of root seeds are needed to advect new pathlines, most of which are saved. A very large number of sibling seeds are released to copy pathlines from these father seeds. The seeding controller adjusts the ratio of root to inheritor seeds to quickly converge to a ratio that enables a nearly constant computational cost per SCAP and accordingly a constant frame rate.

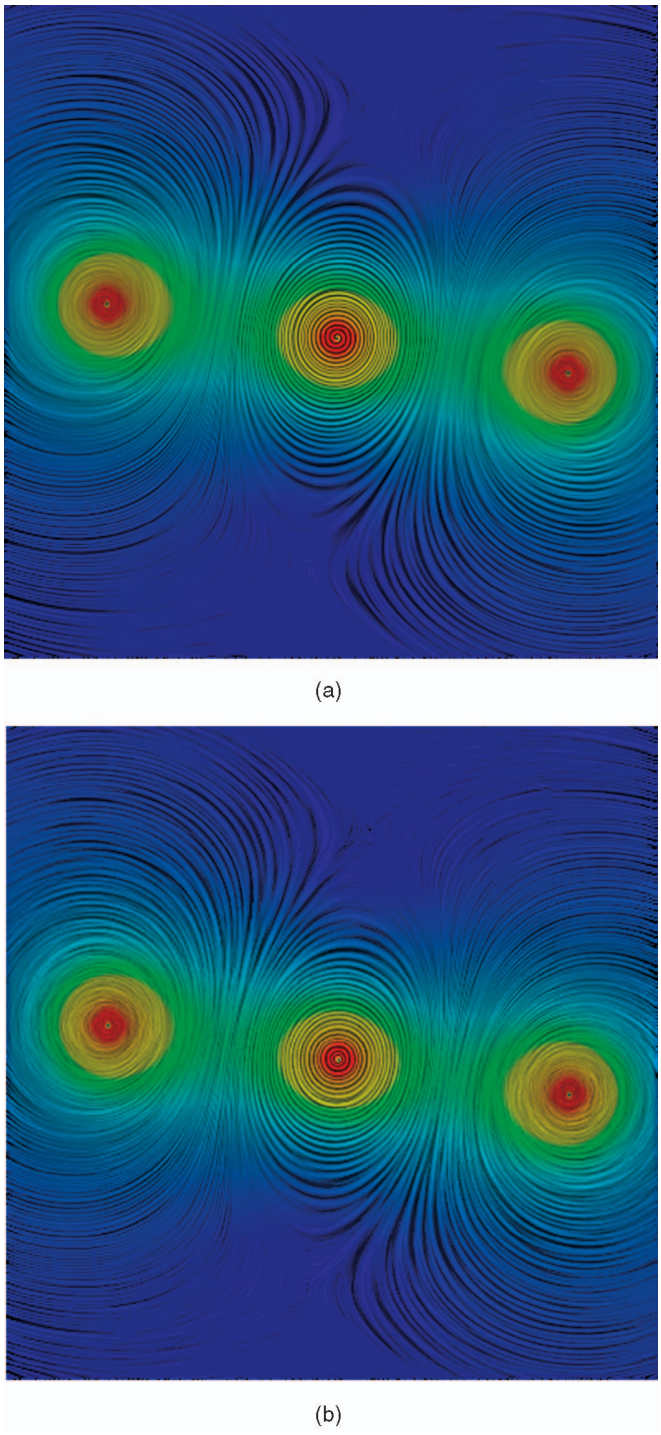
Fig. 5 shows the seed distribution maps in two consecutive SCAPs. The maps use red pixels for root seeds, green pixels for inheritor seeds, and blue pixels for sibling seeds to illustrate the percentages of the pathlines that are obtained by advection, reuse, and copying, respectively, in an SCAP. Note that the distribution patterns just reflect the flow structures. A dense scattering coverage can be maintained by scattering 70 percent of the particles with only 10~15 percent of the pathline integrations required in the original UFLIC algorithm.

3.3 Save, Reuse, and Copy Pathlines

To save, reuse, and copy a pathline for value scattering, AUFLIC uses a box kernel and an RK4-ASSEC pathline integrator that generates evenly sampled (by *SAMPLEN*) points using cubic Hermite interpolation. Since AUFLIC ignores the fading effect [31] used in UFLIC, the contribution a seed with texture value *tex* makes to any downstream sample pixel at time t is:

$$C_{unnormalized} = tex \times \text{SAMPLEN}$$

that is accumulated in the sample pixel's bucket stamped with $\lfloor t \rfloor$ and the weight is:



(a)

(b)

Fig. 9. Two images generated by (a) UFLIC and (b) AUFLIC from the same vortex data (397×397 data points).

$$W = 1/(hits \times \text{SAMPLEN}),$$

where $hits$ is the total number of times the bucket receives values. Since SAMPLEN is omitted through normalization, we can just accumulate tex to the bucket and increase the hit counter. To scatter the texture value of a seed, we only need to know the seed pixel's index, or its coordinate when texture interpolation is used, and the receiver pixel's index.

Pathline reuse involves iterative storage and access, so we use a pixel ring-buffer attached to a header (Fig. 6) to save a pathline that is then inserted into a global pathline list. The

TABLE 1
Component Timings Comparing AUFLIC and UFLIC for Generating All Frames of the Flow Data Sets

Unsteady flow data sets		Weather data set (576 × 291, 41 time steps)		Vortex data set (397 ×397, 101 time steps)	
Algorithms		UFLIC**	AUFLIC	UFLIC**	AUFLIC
Time (sec.s)	Data loading	1.31	1.34	3.20	3.17
	Value scattering	344.94	34.82	863.38	78.50
	Convolution	1.92	1.78	4.32	4.62
	NHP-filtering*	1.17	1.22	2.73	2.82
	Color&output	1.73	1.67	3.91	3.80
	Total	351.07	40.83	877.54	92.91
Acceleration ratio		8.60		9.45	

* NHP-filtering: Noise-jittered High-Pass filtering.
** The UFLIC time breakdowns shown in [33] for the same data sets are different due to different compiler optimization switches. However, the acceleration ratio reported in [33] for UFLIC versus AUFLIC does not change when optimization switches are used in both.

ring-buffer size (*SIZE*) is set as the maximum number of sample points along a pathline in its life span. Initially, the first pathlet is ignored and only the remaining pathlets are sequentially saved in the ring-buffer that records sample pixel indices ranging from 0 to $XRES \times YRES$, the resolution of the field. To reuse the pathline, the son seed changes its role to an inheritor seed and is released to extend the pathline for a new pathlet. To save the pathline, the header field is simply shifted and rewritten to cut off the first pathlet, free the cells, and allocate some unused cells after the end cell for concatenating the new pathlet. The pixel ring-buffer is thus accessed cyclically.

Brute-force value scattering along a pathline begins with releasing the father seed to scatter the texture value. Then, each of the unblocked sibling seeds is released to scatter its value along the known pathline. For a pathline reused by an inheritor seed, multiple accesses to the pixel ring-buffer may be expensive due to intensive wrap operations. Even for a pathline advected by a root seed, it is tedious to access a dynamic pixel array many times to scatter the sibling seed values. Each of the sibling seeds' experience differs only slightly from the father seed in the first time step of the SCAP. To simplify pathline copying for fast value scattering, we use hybrid value scattering, which needs to access the pixel ring-buffer or dynamic pixel array only once to scatter both the father seed value and the sibling seed values. We first add the values of the unblocked sibling seeds to the father seed value and then scatter it as usual. We correct the error of value scattering by simply undoing the contribution a sibling seed, as a component of the "hybrid seed," makes to its upstream samples in both the accumulation bucket and the hit counter (Fig. 7). Prior to hybrid value scattering, the sibling seeds need to be located and checked to see if they are blocked, which is a very fast process.

4 Results and Discussions

We compared AUFLIC with UFLIC in image quality and computational performance using two unsteady flow data sets. The weather data set has 41 time steps and a resolution of 576×291. The vortex data set has 101 time steps and a resolution of 397×397. The computer used for the experiments was an SGI Onyx2 with four 400MHZ MIPS R12000 processors and 4GB memory.

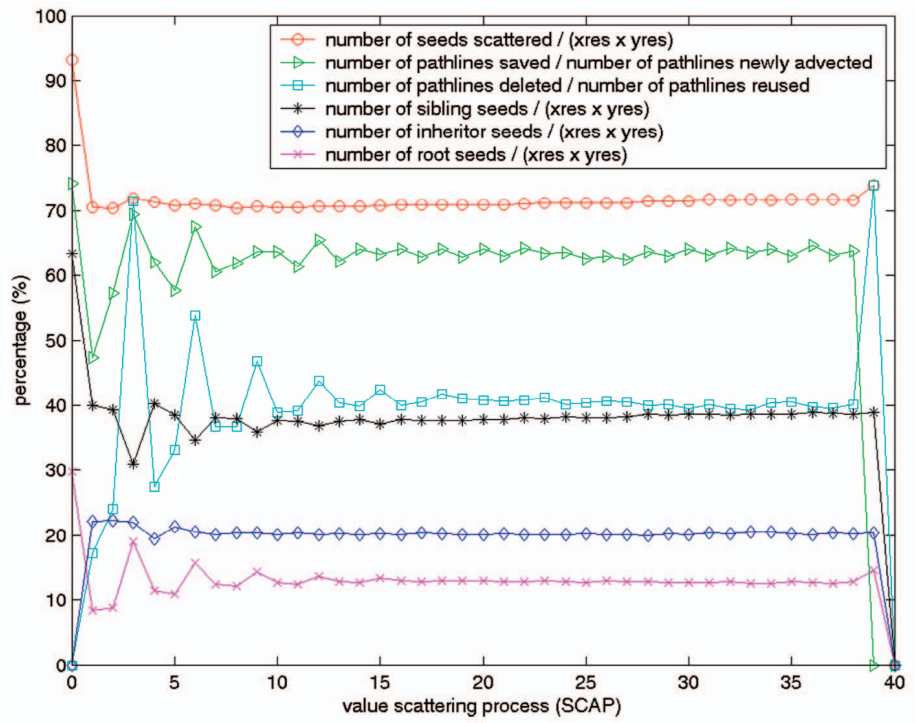
AUFLIC and UFLIC use exactly the same code for data loading, bucket convolution, noise-jittered high-pass filtering, magnitude-based color mapping, and image output. The only difference is in the value scattering process. In both algorithms, original vector magnitudes are scaled and clamped in the data loading stage so that, in any SCAP, the length of a pathline or the number of sample pixels receiving a scattered particle value falls within a fixed range. The maximum value is set as the pixel ring-buffer size in AUFLIC. In all the experiments, the life span is set to four time steps for both UFLIC and AUFLIC. In order to produce 41 frames and 101 frames for the two data sets, respectively, the life span decreases in each of the last four SCAPs by one until it is zero in the last SCAP.

Fig. 8 shows the images produced by UFLIC and AUFLIC from the weather data set. Fig. 9 shows a pair of images generated by the two algorithms from the vortex data set. All the images are color mapped based on the vector magnitude, with blue being lowest and red highest. The images produced by AUFLIC are nearly the same as those produced by UFLIC. No flow features in the UFLIC images are missing in the AUFLIC images and no additional artifacts are introduced in the latter. Even though the fading effect in UFLIC is removed in AUFLIC, the image quality is comparable.

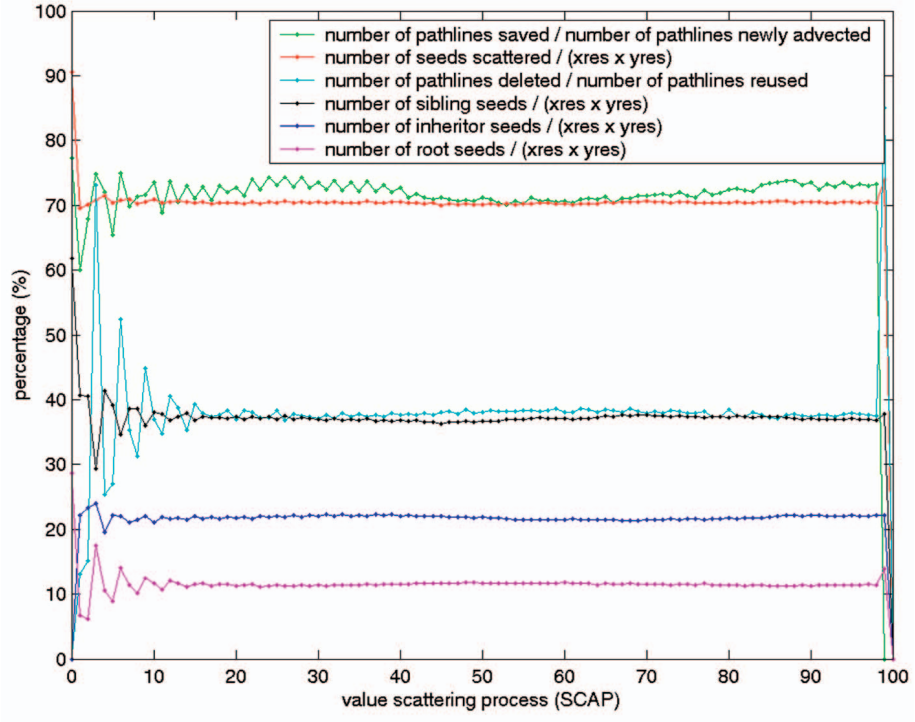
The few differences between the two images are due to different pathlines being traced since UFLIC uses a Euler integrator and AUFLIC uses RK4-ASSEC. Another factor is SAMPLEN, the sampling interval, which may prevent a nonsampled pixel from receiving a scattered value in AUFLIC that it would receive in UFLIC. SAMPLEN is usually less than one pixel and should not be so large that features are missed. In Fig. 8 and Fig. 9, it is set to 2/3.

Table 1 shows the time breakdowns of UFLIC and AUFLIC for visualizing the two data sets. Theoretically, AUFLIC takes the same amount of time as UFLIC in data loading, bucket convolution, noise-jittered high-pass filtering, color mapping, and image output because the same code is used in these stages. As far as value scattering is concerned, AUFLIC consumes much less time than UFLIC due to less pathline integration. AUFLIC generates a frame through the whole pipeline about nine times faster than UFLIC.

Fig. 10 shows the statistics for the seeds and pathlines in the two data sets visualized using AUFLIC. The average



(a)



(b)

Fig. 10. Statistics in visualizing the two data sets using AUFLIC. (a) Weather data set (576×291, 41 time steps). (b) Vortex data set (397×397, 101 time steps).

percentages of inheritor seeds, root seeds, and sibling seeds released each SCAP in the weather flow field are 19.39 percent, 12.96 percent, and 39.36 percent, respectively. In the vortex flow field, the average percentages are 21.37 percent, 11.49 percent, and 37.77 percent. Over 70 percent scattering coverage is achieved with only this small amount of pathline advection. Of the pathlines advected in each SCAP, 62.06 percent and 71.20 percent are saved, respectively, on average in pixel ring-buffers. About 40 percent of the pathlines in the pathline list are deleted in each SCAP

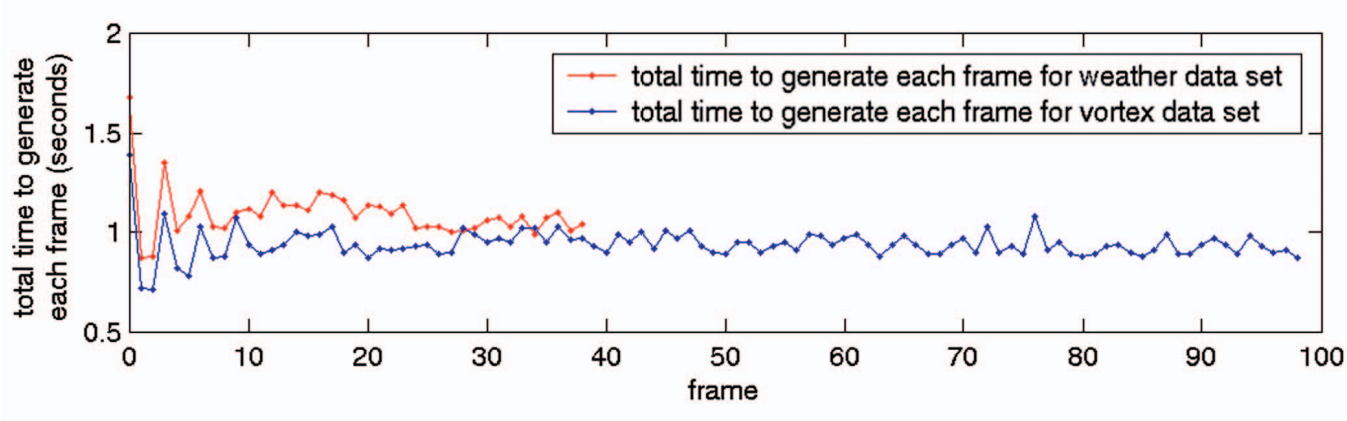


Fig. 11. Total time for AUFLIC to generate each frame (the last two frames omitted due to very small values).

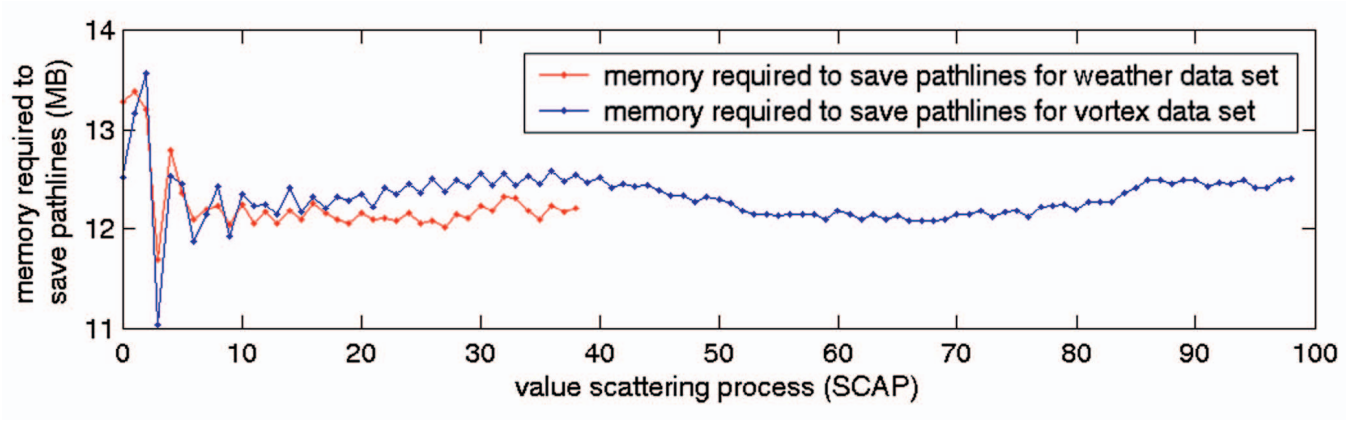


Fig. 12. Memory required for AUFLIC to save pathlines (the last two SCAPs omitted due to very small values).

after being reused. The dynamic seeding controller not only fulfills the flow-driven seeding strategy, but also controls the seed placement for better overall performance. Rapid convergence to an equilibrium is exhibited in the curves in Fig. 10. The fast convergence gives rise to a low-variance frame generation rate (Fig. 11) and a nearly constant memory utilization for saving pathlines (Fig. 12). AUFLIC can be applied to high-resolution unsteady flow visualization because of the small memory footprint.

5 Conclusions and Future Work

We have presented AUFLIC, an accelerated UFLIC algorithm. Based on a flow-driven seeding strategy, AUFLIC releases seeds in the flow direction and, therefore, available pathlines can be copied, saved, and reused to scatter many more particles with fewer pathline integrations in the value scattering process. To implement this strategy, a dynamic seeding controller is designed to intelligently distribute seeds for dense coverage and better overall performance at a very low memory cost. AUFLIC is about 9 times faster than UFLIC with comparable image quality. We were able to visualize an unsteady flow field with 160,000 particles at a near interactive frame rate without additional hardware support.

As for future work, we would like to further enhance AUFLIC by using shorter pathlines or shorter life spans without image quality degradation. A characteristic of the flow-driven seeding strategy is that the output image resolution may be independent of the flow field resolution as is the case in fast LIC [3]. In order to generate a zoomed image (e.g., k times), the same root and inheritor seeds are released in vector space as usual and mapped to image space to locate root and inheritor pixels. (SAMPLEN/k) is used as the sampling interval to scatter k times as many sibling pixels' values by copying the pathline. The pathline is then copied based on the offset in the orthogonal flow direction. The advantage is that the same amount of pathline integration is needed to obtain higher image resolutions.

Acknowledgments

This work was supported by the US Department of Defense HPCMP and US National Science Foundation grant EPS-0132618. The authors would like to thank Dr. Han-Wei Shen for his help with UFLIC questions, implementation details, and valuable suggestions. The unsteady vortex flow data set is courtesy of Dr. Ravi Samtany and Dr. Han-Wei Shen. The authors are grateful to Michael Chupa for helping make the accompanying movies. Thanks also go to the anonymous reviewers for their valuable comments.

References

[1] J.J. van Wijk, "Spot Noise: Texture Synthesis for Data Visualization," *Computer Graphics*, vol. 25, no. 4, pp. 309-318, July 1991.
[2] B. Cabral and L. Leedom, "Imaging Vector Fields Using Line Integral Convolution," *Proc. ACM SIGGRAPH '93*, pp. 263-270, Aug. 1993.

[3] D. Stalling and H.-C. Hege, "Fast and Resolution Independent Line Integral Convolution," *Proc. ACM SIGGRAPH '95,* pp. 249-256, Aug. 1995.
[4] D. Stalling, M. Zockler, and H.-C. Hege, "Parallel Line Integral Convolution," *Proc. First Eurographics Workshop Parallel Graphics and Visualization,* pp. 111-128, Sept. 1996.
[5] L.K. Forssell and S.D. Cohen, "Using Line Integral Convolution for Flow Visualization: Curvilinear Grids, Variable-Speed Animation, and Unsteady Flows," *IEEE Trans. Visualization and Computer Graphics,* vol. 1, no. 2, pp. 133-141, June 1995,
[6] C. Teitzel, R. Grosso, and T. Ertl, "Line Integral Convolution on Triangulated Surfaces," *Proc. Fifth Int'l Conf. in Central Europe Computer Graphics and Visualization (WSCG97),* pp. 572-581, Feb. 1997.
[7] M.-H. Kiu and D.C. Banks, "Multi-Frequency Noise for LIC," *Proc. IEEE Visualization '96,* pp. 121-126, 1996.
[8] R. Wegenkittl, E. Groller, and W. Purgathofer, "Animating Flow Fields: Rendering of Oriented Line Integral Convolution," *Proc. Computer Animation '97,* pp. 15-21, June 1997.
[9] A. Okada and D.L. Kao, "Enhanced Line Integral Convolution with Flow Feature Detection," *Proc. IS & T/SPIE Electronics Imaging '97,* pp. 206-217, Feb. 1997.
[10] H.-W. Shen, C. Johnson, and K.-L. Ma, "Visualizing Vector Fields Using Line Integral Convolution and Dye Advection," *Proc. IEEE Symp. Volume Visualization '96,* pp. 63-70, Oct. 1996.
[11] V. Interrante and C. Grosch, "Strategies for Effectively Visualizing 3D Flow with Volume LIC," *Proc. IEEE Visualization '97,* pp. 421-424, Oct. 1997.
[12] 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 '99,* pp. 233-240, Oct. 1999.
[13] X. Zheng and A. Pang, "HyperLIC," *Proc. IEEE Visualization '03,* pp. 249-256, Oct. 2003.
[14] W. Heidrich, R. Westermann, H.-P. Seidel, and T. Ertl, "Applications of Pixel Textures in Visualization and Realistic Image Synthesis," *Proc. ACM Symp. Interactive 3D Graphics,* pp. 127-134, Apr. 1999.
[15] T. Preuber and M. Rumpf, "Anisotropic Nonlinear Diffusion in Flow Visualization," *Proc. IEEE Visualization '99,* pp. 325-332, Oct. 1999.
[16] D. Burkle, T. Preuber, and M. Rumpf, "Transport and Anisotropic Diffusion in Time-Dependent Flow Visualization," *Proc. IEEE Visualization '01,* pp. 61-67, Oct. 2001.
[17] D.A. Lane, "Scientific Visualization of Large-Scale Unsteady Fluid Flows," *Scientific Visualization,* G.M. Nielson, H. Hagen, and H. Muller, eds., pp. 125-145, Los Alamitos, Calif.: IEEE CS Press, 1997.
[18] B.G. Becker, D.A. Lane, and N.L. Max, "Unsteady Flow Volumes," *Proc. IEEE Visualization '95,* pp. 329-335, 1995.
[19] N. Max and B. Becker, "Flow Visualization Using Moving Textures," *Data Visualization Techniques,* C. Bajaj, ed., pp. 99-105, John Wiley and Sons Ltd., 1999.
[20] V. Verma, D. Kao, and A. Pang, "PLIC: Bridging the Gap between Streamlines and LIC," *Proc. IEEE Visualization '99,* pp. 341-348, Oct. 1999.
[21] B. Jobard, G. Erlebacher, and M.Y. Hussaini, "Hardware-Assisted Texture Advection for Unsteady Flow Visualization," *Proc. IEEE Visualization '00,* pp. 155-162, Oct. 2000.
[22] B. Jobard, G. Erlebacher, and M.Y. Hussaini, "Tiled Hardware-Accelerated Texture Advection for Unsteady Flow Visualization," *Proc. 10th Int'l Conf. Computer Graphics & Vision (Graphicon 2000),* pp. 189-196, 2000.
[23] 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. Sixth Int'l Fall Workshop Vision, Modeling, and Visualization (VMV 2001),* pp. 439-446, Nov. 2001.
[24] J.J. van Wijk, "Image Based Flow Visualization," *Proc. ACM SIGGRAPH '02,* pp. 745-754, July 2002.
[25] J.J. van Wijk, "Image Based Flow Visualization for Curved Surfaces," *Proc. IEEE Visualization '03,* pp. 123-130, Oct. 2003.
[26] A. Telea and J.J. van Wijk, "3D IBFV: Hardware-Accelerated 3D Flow Visualization," *Proc. IEEE Visualization '03,* pp. 233-240, Oct. 2003.
[27] B. Jobard, G. Erlebacher, and M.Y. Hussaini, "Lagrangian-Eulerian Advection of Noise and Dye Textures for Unsteady Flow Visualization," *IEEE Trans. Visualization and Computer Graphics,* vol. 8, no. 3, pp. 211-222, July-Sept. 2002.
[28] D. Weiskopf, G. Erlebacher, M. Hopf, and T. Ertl, "Hardware-Accelerated Lagrangian-Eulerian Texture Advection for 2D Flow Visualization," *Proc. Seventh Int'l Fall Workshop Vision, Modeling, and Visualization (VMV 2002),* pp. 77-84, Nov. 2002.
[29] R.S. Laramee, B. Jobard, and H. Hauser, "Image Space Based Visualization of Unsteady Flow on Surfaces," *Proc. IEEE Visualization '03,* pp. 131-138, Oct. 2003.
[30] D. Weiskopf, G. Erlebacher, and T. Ertl, "A Texture-Based Framework for Spacetime-Coherent Visualization of Time-Dependent Vector Fields," *Proc. IEEE Visualization '03,* pp. 107-114, Oct. 2003.
[31] H.-W. Shen and D.L. Kao, "A New Line Integral Convolution Algorithm for Visualizing Time-Varying Flow Fields," *IEEE Trans. Visualization and Computer Graphics,* vol. 4, no. 2, pp. 98-108, Apr.-June 1998.
[32] Z. Liu and R.J. Moorhead II, "Visualizing Time-Varying Three-Dimensional Flow Fields Using Accelerated UFLIC," *Proc. 11th Int'l Symp. Flow Visualization,* pp. 1-10, Aug. 2004.
[33] Z. Liu and R.J. Moorhead II, "AUFLIC: An Accelerated Algorithm for Unsteady Flow Line Integral Convolution," *Proc. VisSym2002, IEEE TCVG/EuroGraphics,* pp. 43-52, May 2002.
[34] H.C. Hege and D. Stalling, "LIC: Acceleration, Animation, and Zoom," *Texture Synthesis with Line Integral Convolution, Course No. 8, Proc. ACM SIGGRAPH '97 Conf.,* pp. 17-49, Aug. 1997.

Zhanping Liu received the PhD degree in computer science (2000), the MS degree in computer science (1997), and the BS degree in mathematics (1992) from Peking University, TianJin Normal University, and NanKai University, respectively, in the People's Republic of China. He is a postdoctoral associate with the Visualization, Analysis, and Imaging Lab in the ERC/GRI at Mississippi State University. Prior to this position, he was a postdoctoral associate with the Micro-CT Lab in the Department of Radiology at the University of Iowa from 2000 to 2001. His research interests include computer graphics and scientific visualization, particularly flow visualization and volume rendering. He is a member of the ACM SIGGRAPH and the IEEE Computer Society.

Robert J. Moorhead II received the PhD degree in electrical and computer engineering and the MSEE degree from North Carolina State University in 1985 and 1982, respectively. He received the BSEE degree summa cum laude and with research honors from Geneva College in 1980. He is the director of the Visualization, Analysis, and Imaging Lab in the GeoResources Institute and a professor of electrical and computer engineering at Mississippi State University. He previously worked as a research staff member in the Imaging Technologies Department at the IBM T.J. Watson Research Center from 1985 to 1988. He has authored more then 95 papers or book chapters on visualization, image processing, and computer communications. He has received funding from NSF, ARPA, ONR, NRL, AFOSR, the Army Waterways Experiment Station (now ERDC), the Naval Oceanographic Office, NASA, Raytheon, CSC, and Logicon. He was the lead conference cochair for the IEEE Visualization '97 Conference, the chair of the IEEE Computer Society's Technical Committee on Visualization and Graphics in 1999 and 2000, and a papers cochair for the IEEE Visualization 2002 and 2003 Conference. He is a senior member of the IEEE.