Journal of Visualization, Vol. 8, No. 3 (2005) 235-244

A Texture-Based Hardware-Independent Technique for Time-Varying Volume Flow Visualization

Liu, Zhanping* and Moorhead II, Robert J.*

* ERC / GeoResources Institute, P. O. Box 9627, Mississippi State University, MS 39762-9627, USA

Received 30 September 2004
Revised 21 January 2005

Abstract: Existing texture-based 3D flow visualization techniques, e.g., volume Line Integral Convolution (LIC), are either limited to steady flows or dependent on special-purpose graphics cards. In this paper we present a texture-based hardware-independent technique for time-varying volume flow visualization. It is based on our Accelerated Unsteady Flow LIC (AUFLIC) algorithm (Liu and Moorhead, 2005), which uses a flow-driven seeding strategy and a dynamic seeding controller to reuse pathlines in the value scattering process to achieve fast time-dependent flow visualization with high temporal-spatial coherence. We extend AUFLIC to 3D scenarios for accelerated generation of volume flow textures. To address occlusion, lack of depth cuing, and poor perception of flow directions within a dense volume, we employ magnitude-based transfer functions and cutting planes in volume rendering to clearly show the flow structure and the flow evolution.

Keywords: texture-based flow visualization, unsteady 3D flow, LIC, UFLIC, volume rendering.

1. Introduction

Texture-based flow visualization techniques are increasingly important in investigating computational fluid dynamics due to their high resolution and due to the decreased mental reconstruction required compared to traditional graphical-primitive based methods such as arrow plots, streamlines, pathlines (Ushijima and Nezu, 2002), streaklines, stream surfaces, and flow volumes (Becker et al., 1995), which achieve either a local, discrete, coarse view or a cluttered image of the flow field. However, their application to 3D unsteady flows is impeded by computational performance, temporal-spatial coherence, and volume rendering of flow textures. In this paper we present a texture-based hardware-independent technique for time-varying volume flow visualization with high temporal-spatial coherence. Magnitude-based transfer functions and cutting planes are adopted in volume rendering to intuitively show flow structures in individual frames and the flow evolution via a smooth animation.

This paper is organized as follows. We first give a brief review of previous work on texture-based flow visualization in section 2. We then present a texture-based hardware-independent technique for time-dependent volume flows in section 3. Some results are given in section 4 to show the effectiveness and efficiency. The paper concludes with a summary and some directions for future work.

2. Previous Work

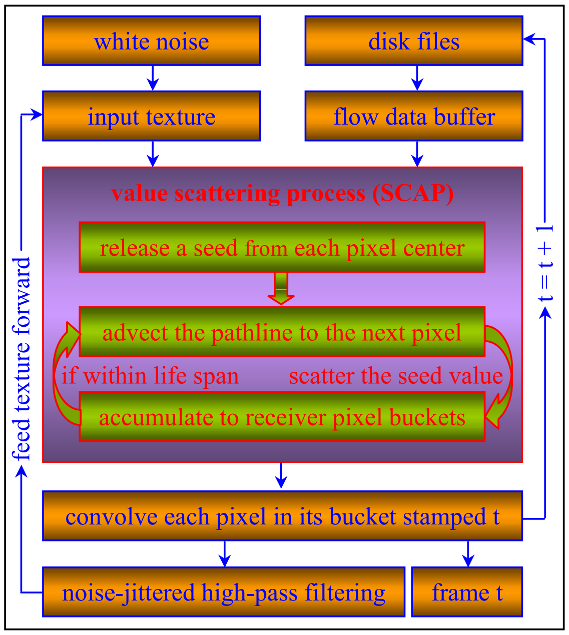
One of the earliest texture-based methods is called spot noise which was presented by Van Wijk (1991) to visualize flow data by stretching and smearing in the flow direction a collection of textured spots to convey the driving flow. Later Cabral and Leedom (1993) proposed another texture synthesis technique, Line Integral

Convolution (LIC), to low-pass filter white noise along pixel-centered, symmetrically bi-directional streamlines to exploit spatial correlation in the flow direction. Analogous to the resulting pattern of wind-blown sand, a LIC image provides a global dense representation of the flow even with high-curvature regions. The last decade has seen numerous optimizations of and extensions to this popular technique such as fast LIC (Stalling and Hege, 1995), curvilinear-grid LIC (Forssell and Cohen, 1995), parallel LIC (Stalling et al., 1996), multi-frequency LIC (Kiu and Banks, 1996), dyed LIC (Shen et al., 1996), oriented LIC (Wegenkittl et al., 1997), triangulated-surface LIC (Teitzel et al., 1997), enhanced LIC (Okada and Kao, 1997), volume LIC (Interrante and Grosch, 1997, Rezk-Salama et al., 1999), and HyperLIC (Zheng and Pang, 2003). LIC was even implemented in hardware by incorporating indirect pixel-texture addressing, i.e., dependent texturing, and texture blending to accelerate streamline advection and texture convolution (Heidrich et al., 1999).

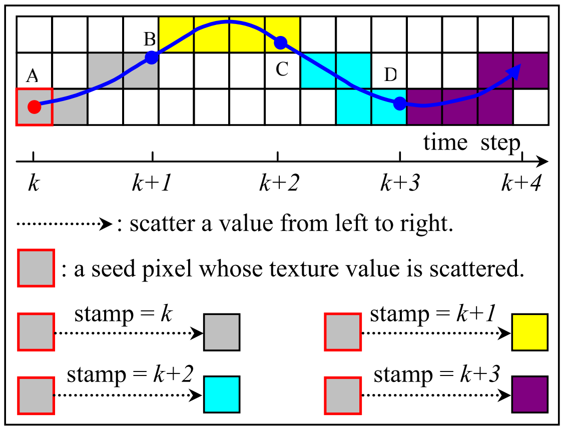
In recent years many texture-based methods have been presented to visualize time-varying flows. Max and Becker (1999) addressed this issue by employing texture mapping coupled with either forward warping of the mesh or backward advection of texture coordinates. Forssell and Cohen (1995) used pathlines as convolution paths in an early effort to adapt LIC for unsteady flows. Verma et al. (1999) proposed Pseudo LIC (PLIC) to simulate a dense representation of time-dependent flows by mapping template textures onto sparse pathline ribbons. Some emerging approaches leverage hardware features to achieve high performance visualization. Jobard et al. (2000) revisited dependent texturing and developed an efficient rendering pipeline for fast texture and dye advection. Weiskopf et al. (2001) took advantage of pixel-texture capabilities to visualize time-varying volume flows. Van Wijk (2002) presented IBFV, which can emulate many visualization techniques at interactive frame rates by cyclically blending a sequence of temporally-spatially low-pass filtered noise textures, one per time, with an iteratively advected image (which is initialized as a black rectangle). IBFV was then extended to visualize curved-surface flows (Van Wijk, 2003) and volume flows (Telea and Van Wijk, 2003). Independent of hardware acceleration (Weiskopf et al., 2002), the LEA algorithm by Jobard et al. (2002) employs backward integration to search the previous frame for the contributing particle, which scatters the texture value to the target pixel of the current frame. Temporal coherence along pathlines is created using successive texture blending while spatial coherence is improved using short-length directional low-pass filtering. Recently Laramee et al. (2003) combined IBFV and LEA to visualize triangulated-surface flows by projecting the surface geometry to image space where backward mesh advection and successive texture blending are performed. Weiskopf et al. (2003) proposed UFAC, which establishes temporal coherence by property advection along pathlines while building spatial correlation by texture convolution along instantaneous streamlines.

Among the most effective algorithms, Unsteady Flow LIC (UFLIC) proposed by Shen and Kao (1998) employs a time-accurate value scattering scheme and a successive texture feed-forward strategy in the visualization pipeline (Fig. 1(a)) to achieve very high temporal-spatial coherence. At each time step, a value scattering process (*SCAP*, Fig. 1(b)) begins by releasing a seed from each pixel center and ranges through several time steps (i.e., *life span*), in which a pathline is advected for the seed to scatter its texture value to the downstream pixels along the trace. The values collected by each pixel at a time step are accumulated and convolved in the stamped bucket to synthesize the corresponding frame, which, after noise-jittered high-pass filtering, is forwarded as the input texture to the next SCAP. The inconsistency between temporal and spatial patterns in IBFV (Van Wijk, 2002), LEA (Jobard et al., 2002), and UFAC (Weiskopf et al., 2003) is successfully resolved by scattering fed-forward texture values in UFLIC. To address the low performance of UFLIC, Liu and Moorhead (2005) presented Accelerated UFLIC (AUFLIC), which achieves an order-of-magnitude speedup over UFLIC by reusing pathlines in the value scattering process.

There have been many texture-based techniques for visualizing curved-surface flows (Forssell and Cohen, 1995, Teitzel et al., 1997, Van Wijk, 2003, Laramee et al., 2003, Mao et al., 1997) and volume flows. However, existing texture-based volume flow visualization methods are either limited to steady flows, e.g., volume LIC (Shen et al., 1996, Interrante and Grosch, 1997, Interrante and Grosch, 1998, Rezk-Salama et al., 1999) and 3D IBFV (Telea and Van Wijk, 2003), or are dependent on special-purpose graphics cards, e.g., hardware-accelerated texture advection (Weiskopf et al., 2001). There has not been yet a hardware-independent solution to texture-based unsteady volume flow visualization.


white noise
disk files
input texture
flow data buffer
value scattering process (SCAP)
release a seed from each pixel center
advect the pathline to the next pixel
if within life span
scatter the seed value
accumulate to receiver pixel buckets
convolve each pixel in its bucket stamped t
noise-jittered high-pass filtering
frame t
feed texture forward
t = t + 1

(a) The UFLIC pipeline.


A
B
C
D
time step
k
k+1
k+2
k+3
k+4
: scatter a value from left to right.
: a seed pixel whose texture value is scattered.
stamp = k
stamp = k+1
stamp = k+2
stamp = k+3

(b) The value scattering process (SCAP). 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*. In the life span, the seed texture value is scattered to the succeeding pixels downstream along the pathline. 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. 1 The UFLIC pipeline and the value scattering process.

3. Visualizing Time-Varying Volume Flows Using AUFLIC

Now we present Accelerated UFLIC (AUFLIC) for visualizing time-varying volume flows, which is described in 2D scenarios for simplicity and clarity, before explaining the 3D extension. We first give the basic idea. Next we introduce the two components, i.e., a flow-driven seeding strategy and a dynamic seeding controller. Magnitude-based transfer functions and cutting planes are then adopted in volume rendering to show 3D AUFLIC textures.

3.1 AUFLIC

The low computational performance of UFLIC lies in the time-accurate value-scattering process, the bottleneck of the pipeline. Specifically, a large amount of Euler pathline integration is computationally expensive due to intensive temporal-spatial vector interpolation and line segment clamping against pixels. In order to obtain a dense value scattering coverage to exploit sufficient temporal-spatial coherence, UFLIC uses a conservative seeding scheme by which a seed is always released from each pixel center at the first integer time step of each SCAP. Such a regular seeding pattern does not take into account flow structures and particles are unnecessarily dense in regions of convergence. A pathline is always advected from scratch and is deleted when the life span expires. This simplistic use of pathlines ignores intra-SCAP and inter-SCAP correlations, which causes severe pathline redundancy. AUFLIC uses a fourth-order Runge-Kutta integrator with adaptive step size and error control in combination with cubic Hermite polynomial curve interpolation (Hege and Stalling, 1997) to achieve faster and more accurate pathline advection and faster line convolution (simplified to averaging for evenly sampled curves) than the Euler method. The temporally-spatially flexible seeding scheme used in AUFLIC allows for a flow-aligned seed placement, which substantially reduces pathline integration by copying and truncating pathlines in the same SCAP as well as reusing and extending pathlines between SCAPs. The dynamic seeding controller effectively implements the flow-driven seeding strategy to achieve dense value scattering at a low computational cost by adaptively determining whether to advect, copy, or reuse a pathline for a given pixel in a SCAP. Table 1 gives a comparison of UFLIC and AUFLIC.

Table 1. Comparison of UFLIC and AUFLIC.

Items	Methods	UFLIC	AUFLIC
Pathline Integration	pathline integrator	Euler (first-order)	fourth-order Runge-Kutta (RK4)
	step size	line-segment clamp against pixels	adaptive step size
	error control	none	embedded Runge-Kutta formulae
	numerical accuracy	first-order	second-order *
	overall efficiency	slow and inaccurate	fast and accurate
Flow-Driven Seeding Strategy	temporal flexibility	none (always at integer time steps)	within a fraction past a time step
	spatial flexibility	none (always from a pixel center)	an arbitrary position within a pixel
	flow-aligned	no (a lattice pattern)	yes (release seeds along pathlines)
SCAP Correlation	intra-SCAP	ignored	copy and truncate pathlines
	inter-SCAP	ignored	reuse and extend pathlines
Dynamic Seeeding Controller	seeding order	left-to-right, top-down	roughly left-to-right, top-down
	flow structure	ignored	adapt seeding to flow structures
	dense coverage	yes	yes
Line Convolution	evenly sampled	no (non-equidistantly sampled)	yes (cubic hermite polynomial)
	accumulation	weighted by segment length	averaged by the hit number
Result	pathline integration	a large amount	substantially reduced
	performance	low	high (1 order-of-magnitude faster)
	quality	high temporal-spatial coherence	high temporal-spatial coherence

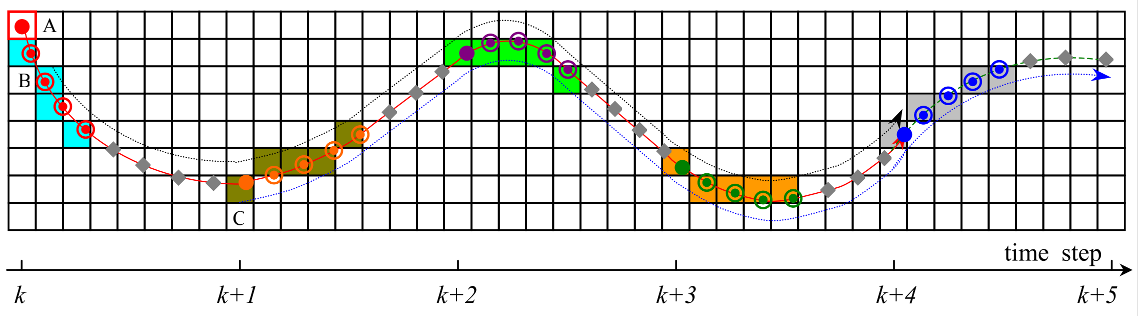
*: As a multi-stage integrator, the RK4 method achieves only second-order accuracy when temporal interpolation is based on the assumption that the flow varies linearly between two consecutive time steps (Lane, 1997).

3.2 Flow-Driven Seeding Strategy

The flow-driven seeding strategy is based on temporal and spatial seeding flexibilities, by which seed distribution is slightly de-coupled from pixel-based image space and strict frame timing, but instead is geared toward physical space where the flow structure is exhibited by flow-aligned particles and toward slightly more continuous seed release than the intermittent mode used in UFLIC. Spatial flexibility allows a seed to be released at an arbitrary position within a pixel rather than exactly at the pixel center. Temporal flexibility allows a seed to be released within a small fractional time past the beginning of a time step (i.e., the first time step of a SCAP) rather than strictly at the very beginning point of the time step. Given the two flexibilities, seeds are released along known pathlines, if available, 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 subsequently released at several positions downstream along the same 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 though 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. Fig. 2 illustrates the flow-driven seeding strategy.

3.3 Dynamic Seeding Controller

To implement the flow-driven seeding strategy, AUFLIC adopts a dynamic seeding controller to govern the seed distribution in a SCAP to determine for each pixel whether a pathline is advected from scratch, reused and extended, copied and truncated, saved for the next SCAP, or deleted. Without an effective controller, many seeds might be released from the same pixel in a SCAP as more and more pathlines are copied and reused while there might be no seeds released from other pixels in regions of divergence. The dynamic seeding controller is used to provide an adaptive, global, and organized control over seed placement that prevents redundant pathline copying or reuse while maintaining a dense value scattering coverage. 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. The dynamic seeding controller addresses this problem by balancing pathline reuse and advection in each SCAP and a nearly constant frame rate is therefore achieved for optimal overall performance. The controller works by dynamically updating two seeding states, *CurSema* (for the current SCAP) and *NxtSema* (for the next SCAP), for each pixel that either allow for a seed release (if the pixel is *open*, i.e., there has not yet been a seed release) or block further releases (if the pixel is *closed*, i.e., there has been already a seed release) to ensure no more than one seed is released from the same pixel in a SCAP. The CurSema determines whether a pathline is advected, copied, or reused in the current SCAP while the NxtSema indicates whether a pathline in the current SCAP is saved for the next SCAP or deleted. When a SCAP begins, the CurSema array is first refreshed with the NxtSema array and the latter is opened. Then the two arrays are dynamically updated as the SCAP runs. Fig. 3 illustrates the dynamic seeding controller.



ξ : an arbitrary position within a pixel. δ: a small fractional time past a time step.

◆ a regular sample point to receive scattered seed values. Life span = 4 time steps.

🟥 a seed released from a pixel center at time step *k* in SCAP *k* to advect a pathline from scratch.

◉ a seed released from ξ at $k+\delta$ in SCAP *k* along the known pathline to copy and truncate the pathline.

● ● ● ● seeds released from ξ at $k+1+\delta$, $k+2+\delta$, $k+3+\delta$, $k+4+\delta$ in SCAP *k+1*, *k+2*, *k+3*, *k+4* along the known pathline saved in SCAP *k, k+1, k+2, k+3,* respectively, to reuse and extend it.

◉ ◉ ◉ ◉ seeds released from ξ at $k+1+\delta$, $k+2+\delta$, $k+3+\delta$, $k+4+\delta$ in SCAP *k+1*, *k+2*, *k+3*, *k+4* along the known pathline reused in SCAP *k+1, k+2, k+3, k+4,* respectively, to copy and truncate it.

┈┈➤ the pathline (offsetted) of seed B in SCAP *k*, obtained by copying and truncating that of seed A.

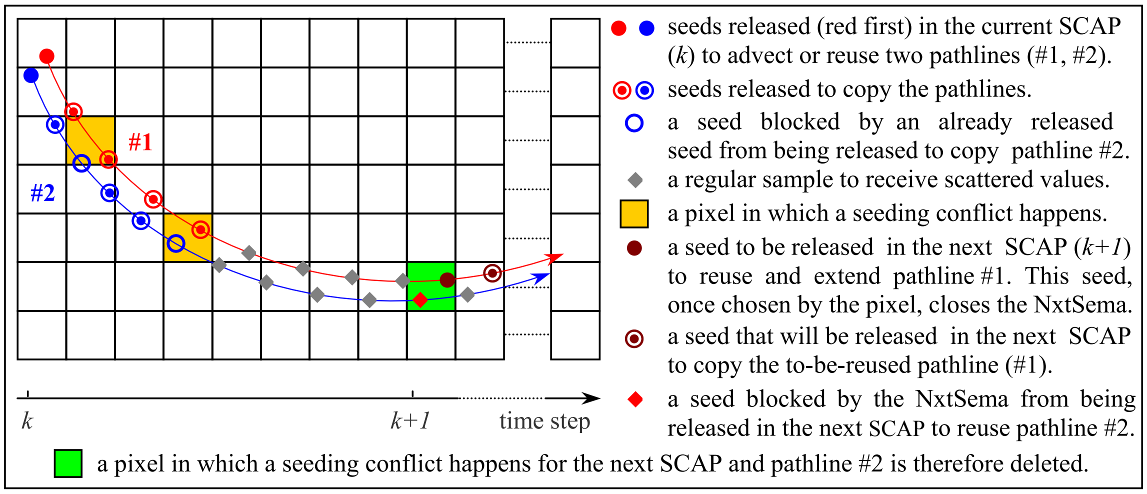
┈┈➤ the pathline (offsetted) of seed C in SCAP *k+1*, obtained by reusing and extending that of seed A.

- - - - - - the extended part of the pathline that is advected in SCAP *k+1* by seed C.

🟦 🟫 🟩 🟧 ⬜ pixels in which seeds may neither be released exactly at pixel centers nor exactly at time step *k*, *k+1*, *k+2*, *k+3*, *k+4* in SCAP *k*, *k+1*, *k+2*, *k+3*, *k+4*, respectively, to advect new pathlines.

Fig. 2 The flow-driven seeding strategy. Given the temporal and spatial seeding flexibilities, seeds released along a known pathline can copy and truncate in the same SCAP, as well as reuse and extend between consecutive SCAPs, the pathline for fast value scattering.

Governed by the seeding controller, a SCAP begins by reusing pathlines saved in the previous SCAP (if they are available), followed by scanning image space in left-to-right, top-to-bottom order to advect new pathlines for still open pixels. Pathline copying is applied each time a pathline is reused or advected while several pixels in the opposite flow direction are reserved to enable pathline copying even in right-to-left or bottom-to-top flow areas. The pathline, if blocked, is deleted; otherwise it is saved for use in the next SCAP.



● ● seeds released (red first) in the current SCAP (k) to advect or reuse two pathlines (#1, #2).

◉ ◉ seeds released to copy the pathlines.

○ a seed blocked by an already released seed from being released to copy pathline #2.

◆ a regular sample to receive scattered values.

🟨 a pixel in which a seeding conflict happens.

● a seed to be released in the next SCAP ($k+1$) to reuse and extend pathline #1. This seed, once chosen by the pixel, closes the NxtSema.

◉ a seed that will be released in the next SCAP to copy the to-be-reused pathline (#1).

◆ a seed blocked by the NxtSema from being released in the next SCAP to reuse pathline #2.

🟩 a pixel in which a seeding conflict happens for the next SCAP and pathline #2 is therefore deleted.

Fig. 3 The seeding controller dynamically updates two states for each pixel. The current state determines whether a seed is released (if the pixel is open) or blocked (if the pixel is closed) and the next state determines whether a pathline is saved (e.g., the red pathline) for the next SCAP or deleted (e.g., the blue pathline).

3.4 Rendering Volume Flow Textures

The extension of AUFLIC to volume flows is straightforward. The flow-driven seeding strategy and the dynamic seeding controller work the same way for volume flows as for 2D scenarios. The small memory footprint of AUFLIC allows it to be used for large unsteady volume flow visualization. Solid white noise is typically used as the initial input texture. Sparse noise used in volume LIC (Interrante and Grosch, 1998) for steady flow visualization may not be applied in 3D AUFLIC for time-varying flows since the successive texture feed-forward strategy hurts the effectiveness of the initial sparse noise and allows for no further input.

Volumetric textures generated by 3D AUFLIC need to be displayed using volume rendering. Volume rendering techniques such as ray casting (Levoy, 1988) eliminate the construction of intermediate geometric representations that are needed in iso-surface algorithms like marching cube (Lorensen and Cline, 1987), instead operate directly on voxels by using a light absorption-transmission model and a transfer function to assign colors and opacities to voxels that are composited along the view direction. Volume rendering has been successfully used in medical data visualization, but volume rendering of flow textures, e.g., LIC volumes, is still an open problem due to the daunting task of designing appropriate transfer functions. Without an effective transfer function, flow directions and interior structures cannot be revealed to provide an insight into the dense volume.

Although 3D AUFLIC texture values convey temporal-spatial correlation along flow lines, they can only be used to compute gradients as normals in the illumination stage to determine *grayscale* voxel intensities. They offer no useful guidance for transfer function design due to lack of intrinsic physical information (e.g., provided by a CT data volume) that can be exploited to distinguish between different components such as bones and soft tissues. Flow directions and depth are indiscernible in a dense cloud-like image volume-rendered using a transfer function based on the flow texture (Fig. 4). As part of the transfer function, the poor color mapping based on irregular texture values largely deteriorates the visual appearance. On the other hand, using a transfer function based on the flow magnitude, as shown in Fig. 5, reveals the interior structure and flow directions, as can be seen in the top-left image in Fig. 6. This image and that in Fig. 4 were rendered from the same texture volume computed using 3D AUFLIC from a time-varying flow dataset of wind evolution in the upper atmosphere. Our volume-rendering model (Fig. 5) is related to but different from the bi-variate volume-rendering model that Shen et al. (1996) presented to render *the mixture of two volume datasets* with *separate* transfer functions. Our method also takes two volume datasets, i.e., flow texture and flow magnitude, as the input, but it renders only the former while the latter is used to define color (strictly, *hue*) and opacity mappings in the transfer function to modulate *grayscale* voxel intensities obtained from the former. The magnitude volume is a by-product of generating the texture volume in 3D AUFLIC and hence no additional computation is required. Although the bi-variate model allows the weighting on the magnitude volume and the LIC volume to vary, the flow patterns are quite obscure. Flow directions and interior structures can be clearly depicted by using our rendering method.

Tuning color and opacity mappings in a transfer function to highlight or suppress certain parts can reveal flow structures. Cutting planes, though simple, are very effective in providing region-of-interest (ROI) selection and insight into interior flow structures by culling occluding sections. Flow structures at an instantaneous time can be explored by shifting arbitrarily-oriented cutting planes through the volume. Animated cutting planes simulate depth cuing. Individual time frames of high temporal coherence can be animated, with optional cutting planes, to investigate the flow evolution. Other interaction techniques include translation, rotation, and zooming.

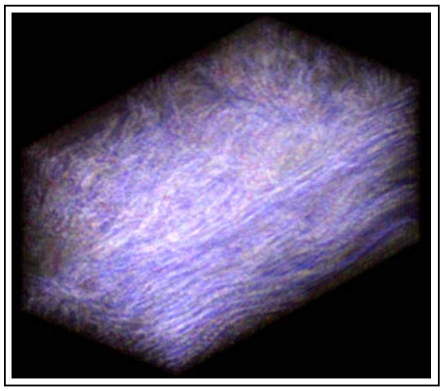


Fig. 4 Rendering of a wind data volume by a texture-based transfer function.

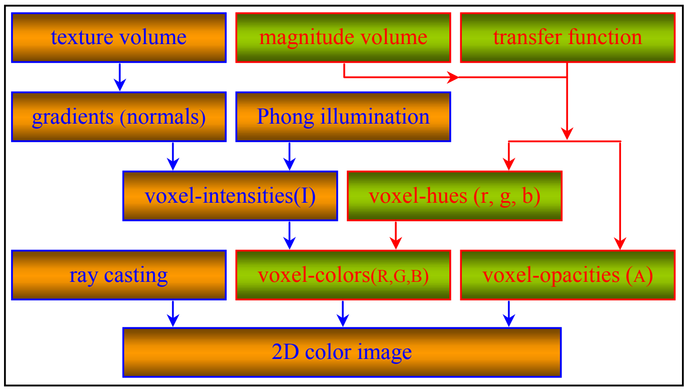


Fig. 5 Our pipeline for rendering 3D AUFLIC textures using a magnitude-based transfer function.

4. Results

We implemented 3D UFLIC and 3D AUFLIC using Visual C++ on Windows XP / Dell Inspiron 2650 notebook (Pentium IV 1.70 GHz, 512MB RAM). A time-varying flow dataset (144 × 73 × 81, 41 time steps) of wind evolution in the upper atmosphere was used for the test. Table 2 shows the time breakdown of the two algorithms for generating 41 volume textures. The test shows 3D AUFLIC is nearly 5 times faster than 3D UFLIC.

Table 2. The time breakdown of 3D UFLIC and 3D AUFLIC in comparison (in seconds).

Items \ Methods	3D UFLIC	3D AUFLIC
Data loading	41.57	40.96
Value scattering	2830.47	552.58
Convolution	4.83	4.84
Noise-jittered high pass filtering	19.12	18.83
Texture output	5.01	4.96
Total	2901.01	622.17
Acceleration ratio	4.70	

We also implemented the aforementioned volume rendering utilities to visualize the resulting volume textures at interactive frame rates on the same notebook without a GPU. The images in Fig. 6 were volume-rendered using our magnitude-based transfer functions. The top-left image was generated from the same texture volume as used in Fig. 4, but it is visually pleasing and insightful thanks to the better transfer function. The top-right image was produced from the texture volume of another time step. The volume was rotated and transparency mapping was tuned to highlight flow lines. Interior flow structures are easily discernable in the top two images even though they were rendered without cutting planes. The middle-left image is the snapshot of a volume sliced with an axis-aligned cutting plane while the middle-right image shows in-depth flow structures revealed by using two orthogonal axis-aligned cutting planes. The images in the bottom row are two snapshots of a volume sliced with a shifting view-aligned cutting plane (see http://www.erc.msstate.edu/~zhanping/Research/FlowVis/VAUFLIC/VAUFLIC.htm for more accompanying images and animations).

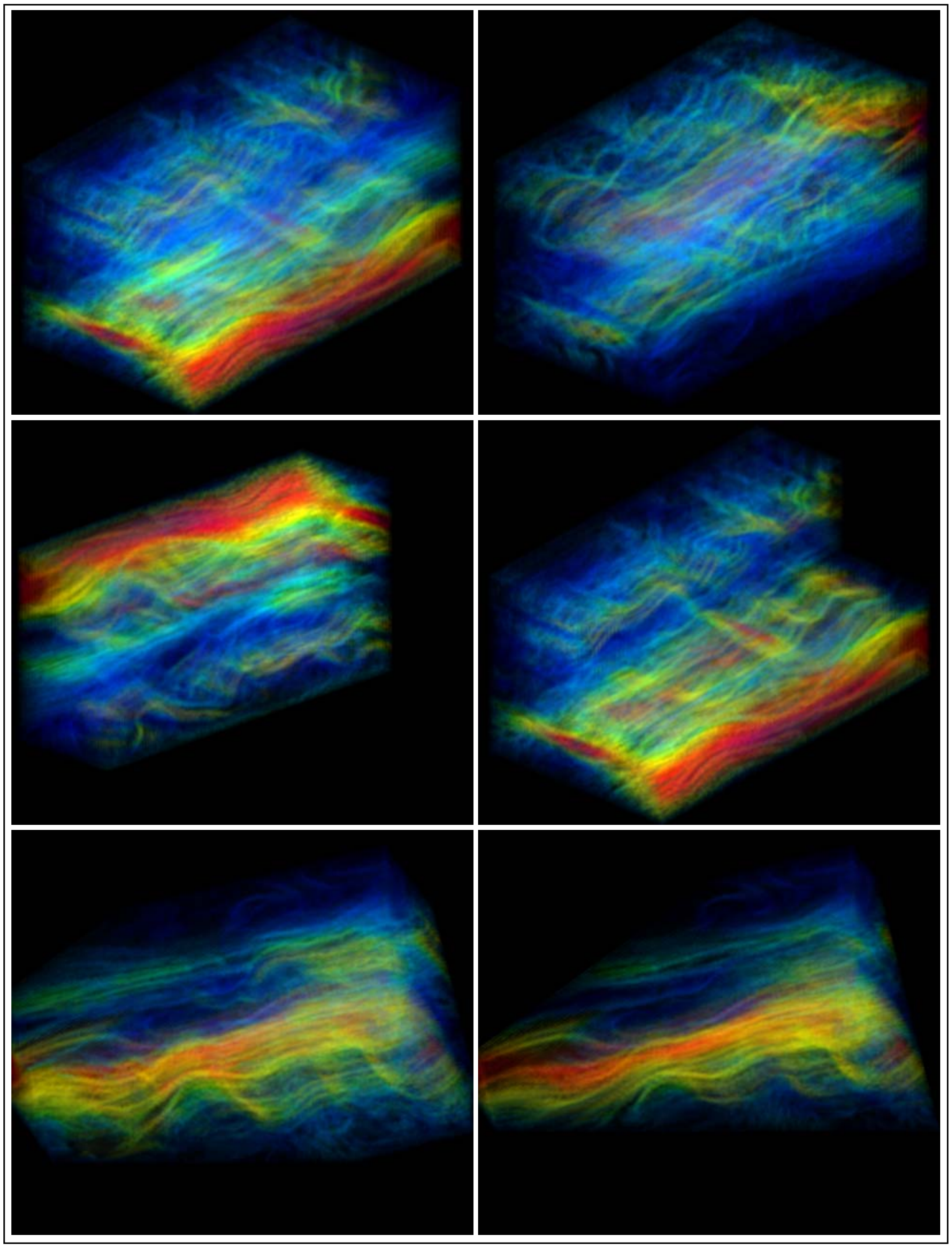


Fig. 6 Images of a time-varying volume flow dataset visualized by using 3D AUFLIC plus volume rendering with magnitude-based transfer functions and cutting planes (top row: without cutting planes; middle row: with axis-aligned cutting planes; bottom row: with view-aligned cutting planes).

5. Conclusions and Future Work

Accelerated UFLIC uses a flow-driven seeding strategy and a dynamic seeding controller to reuse pathlines in the value scattering process to achieve fast unsteady flow visualization with high temporal-spatial coherence. We have extended this texture-based hardware-independent technique for time-varying volume flow visualization and employed magnitude-based transfer functions as well as cutting planes in volume rendering to effectively display flow structures in a dense volume. Future work includes further accelerating AUFLIC by using shorter pathlines while maintaining high temporal-spatial coherence. The rendering of volumetric flow textures may be improved by using visibility-impeding 3D halos (Interrante and Grosch, 1998). Another direction is to enhance ROI selection using a significance map (Chen et al., 2001) that defines for each voxel a value of importance based on flow features (e.g., topological elements).

Acknowledgments

This work was supported by DoD HPCMP and NSF grant EPS-0132618.

References

Becker, B. G., Lane, D. A., and Max, N. L., Unsteady Flow Volumes, IEEE Visualization 95, (1995), 329-335.

Cabral, B. and Leedom, L. C., Imaging Vector Fields Using Line Integral Convolution, SIGGRAPH 93, (1993), 263-270.

Chen, L., Fujishiro, I., and Suzuki, Y., Comprehensible Volume LIC Rendering Based on 3D Significance Map, RIST / TOKYO GeoFEM Report, (2001), GeoFEM.2001-012.

Forssell, L. K. and Cohen, S. D., Using Line Integral Convolution for Flow Visualization: Curvilinear Grids, Variable-Speed Animation, and Unsteady Flows, IEEE TVCG, 1-2 (1995), 133-141.

Hege, H.-C. and Stalling, D., LIC: Acceleration, Animation, and Zoom, Texture Synthesis with Line Integral Convolution (Course #8), SIGGRAPH 97, (1997), 17-49.

Heidrich, W., Westermann, R., Seidel, H.-P., and Ertl, T., Applications of Pixel Textures in Visualization and Realistic Image Synthesis, ACM Symposium on Interactive 3D Graphics, (1999), 127-134.

Interrante, V. and Grosch, C., Strategies for Effectively Visualizing 3D Flow with Volume LIC, IEEE Visualization 97, (1997), 421-424.

Interrante, V. and Grosch, C., Visualizing 3D Flow, IEEE CG & A, 18-4 (1998), 49-53.

Jobard, B., Erlebacher, G., and Hussaini, M. Y., Hardware-Assisted Texture Advection for Unsteady Flow Visualization, IEEE Visualization 00, (2000), 155-162.

Jobard, B., Erlebacher, G., and Hussaini, M. Y., Lagrangian-Eulerian Advection of Noise and Dye Textures for Unsteady Flow Visualization, IEEE TVCG, 8-3 (2002), 211-222.

Kiu, M.-H. and Banks, D. C., Multi-Frequency Noise for LIC, IEEE Visualization 96, (1996), 121-126.

Lane, D. A., Scientific Visualization of Large-Scale Unsteady Fluid Flows, Scientific Visualization (Editors: Nielson, G. M., Hagen, H., and Muller, H.), IEEE Computer Society Press, (1997), 125-145.

Laramee, R. S., Jobard, B., and Hauser, H., Image Space Based Visualization of Unsteady Flow on Surfaces, IEEE Visualization 03, (2003), 131-138.

Levoy, M., Display of Surfaces from Volume Data, IEEE CG & A, 8-5 (1988), 29-37.

Liu, Z. and Moorhead, R. J., Accelerated Unsteady Flow Line Integral Convolution, IEEE TVCG, 11-1 (2005), to appear.

Lorensen, W. E. and Cline H. E., Marching Cubes: A High Resolution 3D Surface Construction Algorithm, Computer Graphics, 21-4 (1987), 163-169.

Mao, X., Kikukawa, M., Fujita, N., and Imamiya, A., Line Integral Convolution for Arbitrary 3D Surfaces Through Solid Texturing, 8th EuroGraphics Workshop on Visualization in Scientific Computing, (1997), 57-69.

Max, N. L. and Becker, B., Flow Visualization Using Moving Textures, Data Visualization Techniques (Editor: Bajaj, C.), John Wiley and Sons Ltd., (1999), 99-105.

Okada, A. and Kao, D. L., Enhanced Line Integral Convolution with Flow Feature Detection, IS & T / SPIE Electronics Imaging 97, 3017 (1997), 206-217.

Rezk-Salama, C., Hastreiter, P., Teitzel, C., and Ertl, T., Interactive Exploration of Volume Line Integral Convolution Based on 3D-Texture Mapping, IEEE Visualization 99, (1999), 233-240.

Shen, H.-W., Johnson, C., and Ma, K.-L., Visualizing Vector Fields using Line Integral Convolution and Dye Advection, IEEE Symposium on Volume Visualization 96, (1996), 63-70.

Shen, H.-W. and Kao, D. L., A New Line Integral Convolution Algorithm for Visualizing Time-Varying Flow Fields. IEEE TVCG, 4-2 (1998), 98-108.

Stalling, D. and Hege, H.-C., Fast and Resolution Independent Line Integral Convolution, SIGGRAPH 95, (1995), 249-256.

Stalling, D., Zockler, M., and Hege, H.-C., Parallel Line Integral Convolution, 1st Eurographics Workshop on Parallel Graphics and Visualization, (1996), 111-128.

Teitzel, C., Grosso, R., and Ertl, T., Line Integral Convolution on Triangulated Surfaces, 5th International Conference in Central Europe on Computer Graphics and Visualization, (1997), 572-581.

Telea, A. and van Wijk, J. J., 3D IBFV: Hardware-Accelerated 3D Flow Visualization, IEEE Visualization 03, (2003), 233-240.

Ushijima, S. and Nezu, I., Parallel Computation and Numerical Visualization for Non-Uniform Particles Included in Gas and Liquid Flows, Journal of Visualization, 5-4 (2002), 327-334.

Van Wijk, J. J., Spot Noise: Texture Synthesis for Data Visualization, Computer Graphics, 25-4 (1991), 309-318.

Van Wijk, J. J., Image Based Flow Visualization, SIGGRAPH 02, (2002), 745-754.

Van Wijk, J. J., Image Based Flow Visualization for Curved Surfaces, IEEE Visualization 03, (2003), 123-130.

Verma, V., Kao, D. L., and Pang, A., PLIC: Bridging the Gap Between Streamlines and LIC, IEEE Visualization 99, (1999), 341-348.

Wegenkittl, R., Groller, E., and Purgathofer, W., Animating Flow Fields: Rendering of Oriented Line Integral Convolution, Computer Animation 97, (1997), 15–21.

Weiskopf, D., Hopf, M., and Ertl, T., Hardware-Accelerated Visualization of Time-Varying 2D and 3D Vector Fields by Texture Advection via Programmable Per-pixel Operations, 6th International Fall Workshop on Vision, Modeling, and Visualization, (2001), 439-446.

Weiskopf, D., Erlebacher, G., Hopf, M., and Ertl, T., Hardware-Accelerated Lagrangian-Eulerian Texture Advection for 2D Flow Visualization, 7th International Fall Workshop on Vision, Modeling, and Visualization, (2002), 77-84.

Weiskopf, D., Erlebacher, G., and Ertl, T., A Texture-Based Framework for Spacetime-Coherent Visualization of Time-Dependent Vector Fields, IEEE Visualization 03, (2003), 107-114.

Zheng, X. and Pang, A., HyperLIC, IEEE Visualization 03, (2003), 249-256.

Author Profile

Zhanping Liu: He is a Research Associate with the Visualization, Analysis, and Imaging Lab in the ERC / GRI at Mississippi State University. Prior to this position, he was a Post-doctoral Associate with the Micro-CT Lab in Department of Radiology at the University of Iowa. He 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. His research interests include computer graphics and scientific visualization, particularly flow visualization and volume rendering. Dr. Liu is a member of the ACM SIGGRAPH and the IEEE Computer Society.

Robert J. Moorhead II: He is Director of the Visualization, Analysis, and Imaging Lab and Professor of Electrical and Computer Engineering at Mississippi State University. He has over 95 publications on visualization, image processing, and computer communications. Professor Moorhead received his PhD and MSEE from North Carolina State University in 1985 and 1982 respectively, and his BSEE from Geneva College in 1980. He was the Conference Chair for the IEEE Visualization 97 Conference, Chair of the IEEE Computer Society's Technical Committee on Visualization and Graphics in 1999 and 2000, and Paper Co-Chair for the IEEE Visualization 2002 and 2003 Conference.