Eurographics/ IEEE-VGTC Symposium on Visualization (2006), pp. 1–6
Thomas Ertl, Ken Joy, and Beatriz Santos (Editors)

GPUFLIC: Interactive and Accurate Dense Visualization of Unsteady Flows

Guo-Shi Li[1] and Xavier Tricoche[1] and Charles Hansen[1]

[1] Scientific Computing and Imaging Institute, University of Utah

Abstract

The paper presents an efficient and accurate implementation of Unsteady Flow LIC (UFLIC) on the Graphics Processing Unit (GPU). We obtain the same, high quality texture representation of unsteady two-dimensional flows as the original, time-consuming method but leverage the features of today's commodity hardware to achieve interactive frame rates. Despite a remarkable number of recent contributions in the field of texture-based visualization of time-dependent vector fields, the present paper is the first to provide a faithful implementation of that prominent technique fully supported by the graphics pipeline.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Pictures/Image Generation

1. Introduction

Texture-based techniques constitute a standard choice to generate intuitive visualizations of steady flow data. The resulting dense representation is a powerful way to convey essential patterns of the vector field while avoiding the tedious task of seeding individual streamlines to capture all the structures of interest. Although it seems natural to extend these methods to obtain animated visualizations of time-dependent flows, this approach is faced with two major challenges. First, it must address the conflicting requirements of ensuring temporal coherence of the computed frames on one hand, and emphasizing the flow structures present in each frame on the other hand. While the former is mandatory to avoid flickering artifacts, the latter is key to an effective interpretation of the flow structures and their evolution over time. The second challenge concerns the intrinsic high computational complexity of these methods, which is dramatically increased by the necessary processing of many time steps.

These limitations have motivated a significant body of research on texture-based flow visualization of unsteady flows. Multiple methods have been designed, offering a variety of visual effects that each correspond to a particular compromise between time and space coherence. Weiskopf et al. [WEE03] proposed a unified formalism in order to analyze the rather implicit spatio-temporal characteristics underlying these schemes. One common feature of the algorithms presented in recent years is their use of the increasing programmability of the consumer graphics hardware to achieve interactive. Yet, in many cases the restrictions imposed by the very specific nature of GPU programming have resulted in visual representations that do not match the quality of the original, offline methods and can be difficult to interpret in the post-processing analysis of numerical simulations.

Among the existing techniques in this field, we argue that Unsteady Flow LIC (UFLIC) [SK97, SK98] provides a unique depiction quality for transient flows which comes along with desirable visual properties that nicely fit the intuition of the observer. The major downside of this algorithm however, is the high computational cost caused by the processing of a very dense set of particles. Moreover this is a computation model that does not map naturally to the texture-based data representation available on the GPU because it requires the scattering of per-pixel properties across textures. An accelerated version called AUFLIC [LM02, LM05] of the original algorithm exists but the corresponding performance is far from interactive. In this paper, we present the first implementation of UFLIC that is both accurate with respect to the original method and is entirely supported by the GPU, allowing interactive visualization of complex flow data. We observe that using their frame-

work Weiskopf et al. [WEE03] already suggested a texture-based method that yields analogous pictures. While their approach constitutes an approximation of the actual UFLIC computation and is potentially slow, the implementation that we propose is faithful, straightforward, and efficient.

The remainder of this paper is organized as follows. Related work is discussed in section 2. In particular, we emphasize GPU-based dense visualization methods for unsteady flows. A detailed explanation of UFLIC and its major features are provided in section 3. We also comment on the challenges raised by an efficient implementation and consider existing solutions. Our new GPU implementation is described in section 4. In section 5 we provide results that demonstrate both the accuracy and the interactivity of the proposed method. Finally, we conclude by pointing out interesting avenues of future work in section 6.

2. Related Work

For a general introduction to texture-based flow visualization methods we refer the reader to the existing surveys, e.g. [LHD*04]. Here we focus our presentation on interactive techniques for time-dependent flows and defer a detailed description of UFLIC to the next section.

The first hardware accelerated dense representation of unsteady flows goes back to papers by Jobard et al. [JEH00, JEH01] who exploit hardware features new at the time to achieve interactive animations by means of texture advection. In [JEH01] and subsequently in [JEH02] they develop their so-called Lagrangian-Eulerian Advection (LEA) approach. The basic idea of their method consists in combining backward pathline integration with the advection of an interpolated texture value fetched at the previous iteration. In that way they advect a noise texture over time while avoiding the creation of gaps. To address the issue of noise replication additional noise is injected at random locations. Finally spatial patterns are visualized by blending the current texture with the previous one. The method leverages optimized GPU operations and is therefore very fast. However the corresponding animations are characterized by a limited space and time coherence with a fairly coarse approximation of the path of particles.

A somewhat similar texture advection scheme called Image Based Flow Visualization (IBFV) was introduced by van Wijk [vW02]. Flow structures are obtained by successive alpha-blending of an advected image with a new texture at each step. The paper provides an analysis of different strategies to choose the spatial and temporal properties of the noise texture. As in the case of LEA, this technique benefits from a direct mapping on the graphics hardware and is very fast. The weaknesses of this technique are relatively short paths and limited contrast, which makes it difficult to identify and track patterns.

Both LEA and IBFV were later extended to the visualization of time-dependent flows defined over curved surfaces embedded in three-dimensional space [LvWJH04]. To do so the original algorithms are applied to the projection of a tangential vector field onto the 2D screen. This projection, supported by the graphics hardware, is updated at interactive frame rates for any change in the camera position. Weiskopf and Ertl use a similar idea but combine computations in physical and screen space to achieve better accuracy [WE04].

A formal framework to understand and analyze the space-time characteristics of various texture-based flow visualization methods was proposed by Weiskopf et al. [WEE03]. There are three basic ingredients to their framework: a space-time domain filled by a continuum of pathlines conveying scalar attributes for time coherence, an additional vector field defined over the space-time domain that determines the one-dimensional domain where convolution is performed, and a convolution kernel that is applied to create spatial coherence. The authors use this formalism to describe the schemes mentioned previously. The complete data structure is implemented using textures to permit GPU encoding. Moreover they introduce a new visualization method called Unsteady Flow Advection-Convolution (UFAC) in which the convolution of the space-time property field takes place in spatial slices with a kernel size controlled by the temporal derivative of the vector field, thus measuring unsteadiness. The original implementation of the framework was based on backward integration to advect the property field which made it computationally expensive. Recently an implementation involving forward particle advection was published by the same authors [WSEE05] who use radial basis functions and a divergence driven particle injection-deletion mechanism to ensure constant particle density over time.

3. UFLIC

Inspired by line integral convolution (LIC) [CL93], Unsteady Flow LIC (UFLIC) by Shen and Kao [SK98] is a dense texture synthesis technique to visualize time-varying vector fields. While it is a very powerful method for visualizing steady vector fields, LIC fails to deliver satisfactory results when applied to time-varying flows. As explained in [SK98] native application of LIC to each time step of a unsteady flow dataset results in strong flickering in the animation because of the lack of time coherence of patterns defined in individual time steps. On the other hand a convolution applied along pathlines as proposed by Forsell and Cohen [FC95] leads to very poor spatial patterns and a cumbersome interpretation.

UFLIC provides an elegant and intuitive solution to these problems. Rather than gathering intensity values from each pixel, a large number of particles are released and advected to scatter their intensity values to pixels lying along their path. To properly capture and establish coherence in time, each pixel records the intensity values of the particles cross-

ing it, along with a time stamp. When averaging the individual contributions at each pixel at time t, only intensity values with time stamps in $[t-\delta t, t]$ are used, where δt is the interval between successive time steps. To establish frame to frame coherence and to maintain dense coverage, a new set of particles derived from the current image are injected at the next time step to iteratively scatter the result generated by the previous construction.

Compared to other methods, UFLIC can generate visualizations of high quality. Many methods relying on backward integration (e.g. [JEH01]) suffer from blurred line patterns, due to diffusion created by successive resampling of the property field [Wei04]. UFLIC, on the other hand, is able to generate highly-coherent, well-defined, and crisp structures. Each particle, along with the intensity value it carries, keeps its identity throughout its entire life span. These values are scattered into individual pixel locations without interpolation. Along with the associated time stamps, the contribution of each particle is explicitly defined in the spatio-temporal domain. As a result, UFLIC is able to generate discerning line patterns of high contrast at different temporal and spatial scales in the visualization.

To address the high computational cost of the original UFLIC algorithm a parallel implementation was described in [SK98]. The linked-list data structure associated with each pixel in the original implementation to store intensity-time pairs is replaced by a circular queue to reduce memory requirements. Each entry in the circular queue, called *bucket*, is used to accumulate all contributions made between successive time steps. The number of the entries in the circular queue is equal to the life span of particles expressed in terms of a number of time steps. Inspired by FastLIC [SH95], AUFLIC proposed by Liu and Moorhead [LM05] further accelerates the scheme by reusing pathline computations performed in previous iterations, and dynamically creating additional pathlines at pixel locations not covered by existing pathlines. Yet, the complexity of the corresponding algorithm prevents interactive performance. Using the GPU implementation of their generic framework Weiskopf et al. suggested an algorithm that mimics UFLIC [WEE03]. However, the restrictions imposed by their data structure limit the ability to properly implement UFLIC's accurate time scattering. The solution they proposed consists of increasing the resolution of the time axis, which accounts for the contributions of multiple particles crossing a single pixel during a coarser time step. Nevertheless, it remains unclear what time resolution is needed to convey the specific properties of a given flow.

4. GPU Implementation

Value depositing and scattering is the most time-consuming stage in UFLIC [LM05]. Most of the computation spent in each iteration, however, is dedicated to scattering values to be used in the *future*. Specifically, if the life span of particles

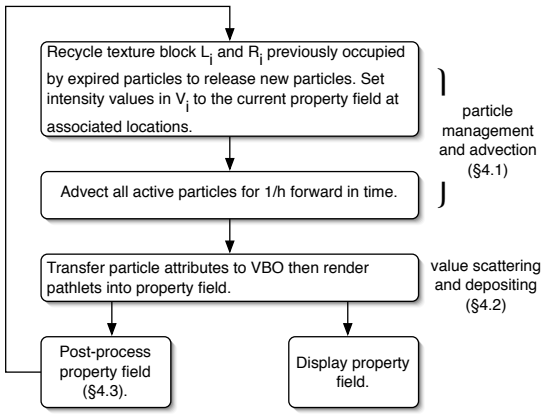

Figure 1: *Illustration of the algorithm.*

.

is N and one image is generated at the end of each time step for animation, only the trajectories made in $1/N$ of the entire life span directly contributes to generating the *current* frame. In UFLIC, the storage of intensity values deposited ahead of the current time requires a ring-bucket data structure for each pixel. Although modern GPUs can provide massive computation power thanks to their *Single Instruction Multiple Data* (SIMD) design, the flexibility of data structures available on GPU is severely restricted by its highly parallel nature. To fully leverage the capability of modern GPUs, we propose a new value scattering scheme for UFLIC that we call "on-the-fly depositing". Instead of tracing particles throughout their entire life span, at every iteration they are advected from their previous locations only up to the current time. The pathline segment in this interval, called *pathlet*, is used to scatter a particle intensity value. Pixels in the domain receive intensities *only* for the current frame; therefore, the need of per-pixel bucket data structure is eliminated. To establish smooth spatial coherence, we generate several animation frames per time step. Particles with a life span greater than zero remain active in subsequent iterations to incrementally generate pathlines with pathlets. At the end of each iteration, a new set of particles is released into the domain with associated intensity values derived from the current property field, while expired particles are removed and recycled. In our algorithm, the entire UFLIC pipeline, mainly particle management and advection, value scattering, and contrast enhancement can be fully executed on the GPU and results in interactive frame rates. Figure 1 illustrates our algorithm. In the following, we detail each step of our implementation.

4.1. Particle Management and Advection

At the beginning of each iteration I, particles are released at every pixel location in a domain of size $M \times N$. Along with the position (x, y), each particle is also associated with an

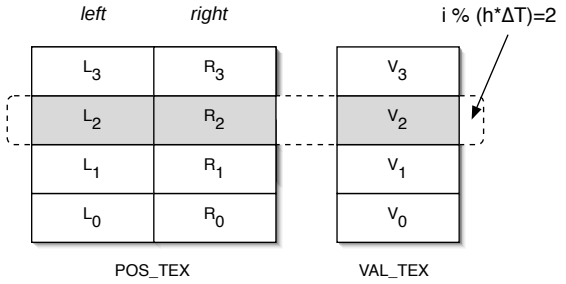

Figure 2: *Texture layout for particle attributes and intensities (shown as $\Delta T \times h = 4$).*

intensity value $\alpha = \mathcal{PF}^{I-1}(x,y)$, where $\mathcal{PF}^{I-1}$ is the property field generated at the previous iteration $I-1$ ($\mathcal{PF}^0$ is initialized to a white noise texture), and also a time stamp ttl, denoting time-to-live, which is initialized to the global life span. The intensity value α of each particle remains unchanged throughout the entire life span.

As discussed previously, excluding the setup and shutdown periods at any given time there are up to $K = \Delta T \times h \times M \times N$ active particles, where ΔT is the global particle life span measured in number of time steps and h is the number of frames per time step. The attributes (x,y) and ttl of each particle are stored in a 32-bit floating point texture of size $(2 \times M, \Delta T \times h \times N)$ named `POS_TEX`. The intensity values α are stored in another 8-bit single channel texture of size $(M, \Delta T \times h \times N)$ named `VAL_TEX`. Please refer to Figure 2. Conceptually, `POS_TEX` is divided into two columns, *left* and *right*. Each column consists of $\Delta T \times h$ blocks, where each block is of size M by N. We denote blocks on *left* and *right* as L_i and R_i. Each pair of L_i and R_i store the attributes which belong to a group of particles released into the domain at iteration i mod $\Delta T \times h$. Alternatively they serve as the input and the output storage of the advection scheme, as will be explained later. After reaching the steady state, at each iteration there is exactly one block pair of particles reaching the life span, hence they are recycled to release new particles. `VAL_TEX` is arranged and recycled in a similar manner with only one column.

We use the fragment shader to perform particle tracing in an efficient manner. At each iteration of advection, a quadrilateral of $M \times \Delta T \times h \times N$ pixels alternatively textured with the left or right half of `POS_TEX` is rendered to generate data streams for subsequent SIMD advection in the fragment shader, which implements higher-order numerical schemes such as Runge-Kutta 4th order. The other half of `POS_TEX` is simultaneously set to the render target using the framebuffer object extension to prevent read/write conflicts. The 2D time-varying vector field is represented as a 3D texture with each slice on the z axis corresponding to a time step [†]. With this construction, the triplet (x,y,t), where t is the global clock indicating the current time, is used as texture coordinates to sample velocities of the time-varying vector field. The advection fragment program outputs the advected particle position and the decremented ttl. When the resulting position exits the domain or ttl is less or equal to zero, it returns an invalid triplet $(-10,-10,-10)$, preventing further advection and value scattering of a given particle.

4.2. Value Scattering and Depositing

To scatter the intensity value of a particle $p(i,j)$, i.e. `VAL_TEX`(i,j), over all the pixels it goes through during the current interval, the pathlet $\mathcal{L}(i,j)$ defined by `POS_TEX`(i, j) and `POS_TEX`(2i, j) (the attribute pair located on *left* and *right*, respectively) is rasterized with color `VAL_TEX`(i,j)*ω into a floating point buffer serving as the property field, where ω is a weighting factor used for time-fading or other adjustments.

To render all pathlets efficiently, we first transfer the content of `POS_TEX` to the vertex storage on GPU memory using the combination of vertex buffer object and pixel buffer object extensions (VBO/PBO) and then draw line segments using proper index values. If a particle has invalid coordinates, hardware clipping will automatically discard the vertex, preventing further depositing. Blending is enabled when rendering pathlets with the blending function set to `(GL_ONE, GL_ONE)` to accumulate incoming intensities and weights at every pixel.

4.3. Postprocessing

After value scattering and depositing, each pixel of the property field contains a tuple $(\sum V * \omega, \sum \omega)$, where $\sum V$ denotes the accumulated intensities. Displaying the property field only requires normalization of the accumulated values at each pixel: $(\sum V * \omega / \sum \omega)$. As pointed out in [SK98], postprocessing the property field is required before it is used to specify the intensities of new particles released in the next iteration in order to maintain satisfying contrast throughout the animation. Specifically, the solution consists in first applying a Laplacian high-pass filter to the rendered image. Special care is needed to ensure that all the entries of the output texture lie in the valid value range.

The second step, called noise jittering, involves the mixing of the high-pass filtered property attribute texture with a noise signal to suppress exceeding high frequencies in the resulting image. In the original method, this noise signal is the same as the one used to initialize the property field. To

[†] Only time steps corresponding to the time interval $1/h$ walked by each particle between two frames need to be present in video memory at a time. If the number of time steps exceeds the texture resolution limit (512 on most architectures currently), time steps can be dynamically inserted into the 3D texture using glTexSubImage3D.

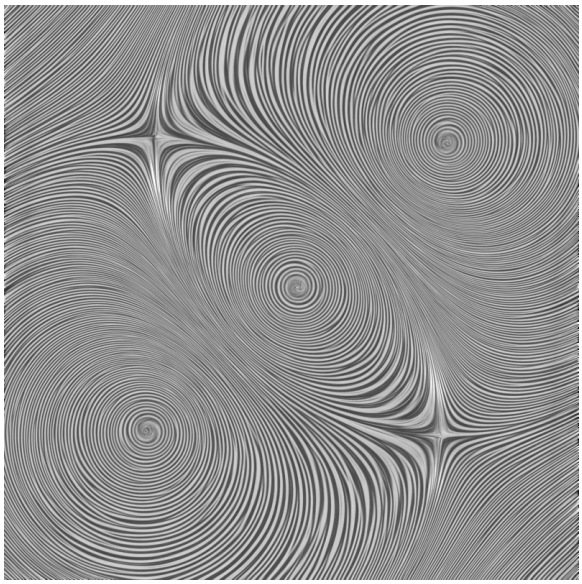

Figure 3: *PSI dataset.*

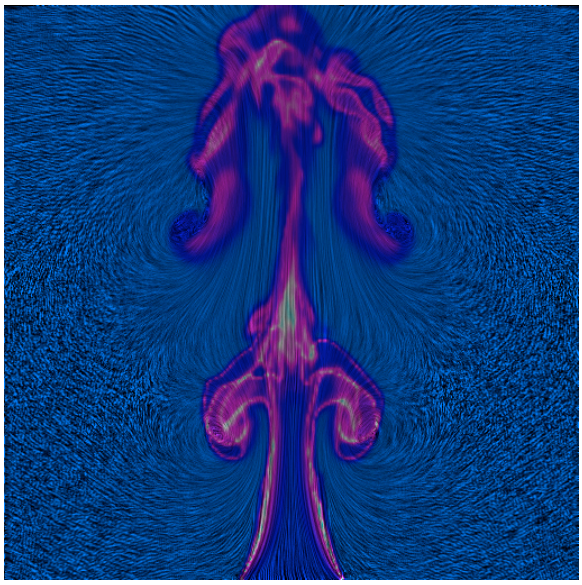

Figure 4: *JP8 dataset.*

avoid frozen patterns in steady regions, we use a periodic noise to provide the expected visual impression of motion along the flow.

5. Results

We implemented the algorithm described in section 4 using C++ and OpenGL on a standard Windows PC. The platform used in our experiments is equipped with an Intel Pentium 4 3.6GHz processor with 2GB of RAM and dual nVidia 7800 GTX cards. All shaders are implemented in a total of roughly 200 lines of `Cg` code. Two time-varying datasets, `PSI`(figure 3) and `JP8`(figure 4) are used in our experiments. `PSI` is a standard synthetic dataset. As Figure 3 shows, our algorithm is able to establish crisp line patterns of high quality, matching those generated in a purely-software, offline manner.`JP8` was results from a large scale numerical simulation of fire and explosion phenomena [CSA]. The color coding in figure 4 highlights the temperature. In this dataset, the variance of velocity magnitude is very large. Strongly established patterns close to the epicenter depict the fast moving nature of explosion, while particles aside are barely moved. Table 1 gives the performance figures in both single and dual GPU (SLI) modes. In most cases we are able to achieve interactive frame rates with different image resolutions using the RK45 numerical scheme.

6. Conclusion

We have presented an implementation of UFLIC fully supported by the GPU that provides the same visual quality as the original technique. One key aspect of our approach consists in a reformulation of the method as an iterative, simultaneous advection of a dense set of particles released at different time steps. Doing so we are able to restrict the computation to the needs of the next frame which significantly improves the efficiency of the algorithm. Moreover we accomplish the value depositing step by drawing line segments thus leveraging an optimized feature of the graphics hardware. These two simple ideas allow for a straightforward implementation that yields high-quality animations of time-dependent flows at interactive frame rate. The performance that we measure on a standard graphics card represents a significant speedup with respect to AUFLIC reported by [LM05], so far the fastest existing implementation of UFLIC to our knowledge.

The contribution of this paper, however, is not to be measured in the number of frames per second of our GPU algorithm. Admittingly, image-based methods that yield even higher frame rates are straightforward to implement, and generalize naturally to arbitrarily complex geometries. In contrast, the key features of our interactive implementation are the accuracy and the unique visual quality that UFLIC provides.

This work suggests several interesting avenues for future research. The first one concerns the improvement of the method for the visualization of flows exhibiting high unsteadiness. This can become necessary to prevent some distracting “ghost” effects to become visible when patterns change very rapidly over time. Another possible extension of our implementation consists in using the particle system that underlies our scheme to carry dye along the flow. We would also like to extend this method to visualize unsteady flows on curved surfaces. This is more challenging than in mesh-

dataset	PSI				JP8			
resolution	256x256		512x512		256x256		512x512	
dual/single GPU	D	S	D	S	D	S	D	S
life span = 2	213.1	182.3	49.2	53.4	96.0	96.2	26.6	21.3
life span = 4	106.3	108.2	10.5	11.2	50.4	53.4	5.25	1.8
life span = 6	64.1	58.1	2.3	1.4	22.8	26.0	2.6	1.4

Table 1: *Performance in FPS. Particle life span in time steps.*

independent, image-based approaches but it will allow us to monitor precisely the evolution of intricate flow structures in this more complex setting.

Acknowledgements

We thank Joe Kniss and Han-Wei Shen for insightful discussions, and James Bigler for assistance with the JP8 dataset. This work was supported by the U.S. Department of Energy through the Center for the Simulation of Accidental Fires and Explosions, under grant W-7405-ENG-48.

References

[CL93] CABRAL B., LEEDOM C.: Imaging vector fields using line integral convolution. In *Proceedings of SIGGRAPH 93* (1993), ACM SIGGRAPH, pp. 263–270.

[CSA] Center for the simulation of accidental fires and explosions. http://www.csafe.utahl.edu.

[FC95] FORSSELL L., COHEN S.: Using line integral convolution for flow visualization: Curvilinear grids, variable-speed animation, and unsteady flows. *IEEE Transactions on Visualization and Computer Graphics 1*, 2 (1995), 133–141.

[JEH00] JOBARD B., ERLEBACHER G., HUSSAINI M. Y.: Hardware-accelerated texture advection for unsteady flow visualization. In *Proceedings of the conference on Visualization '00* (2000), IEEE Computer Society Press, pp. 155–162.

[JEH01] JOBARD B., ERLEBACHER G., HUSSAINI M. Y.: Lagrangian-eulerian advection for unsteady flow visualization. In *Proceedings of the conference on Visualization '01* (Washington, DC, USA, 2001), IEEE Computer Society, pp. 53–60.

[JEH02] JOBARD B., ERLEBACHER G., HUSSAINI M.: Lagrangian-eulerian advection of noise and dye textures for unsteady flow visualization. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (2002), 211–222.

[LHD*04] LARAMEE R., HAUSER H., DOLEISCH H., VROLIJK B., POST F., WEISKOPF D.: The state of the art in visualization: Dense and texture-based techniques. *Computer Graphics Forum 23*, 2 (2004), 143–161.

[LM02] LIU Z., MOORHEAD R. J.: Auflic: An accelerated algorithm for unsteady flow line integral convolution. In *Proceedings of Data Visualization.* (2002).

[LM05] LIU Z., MOORHEAD R. J.: Accelerated unsteady flow line integral convolution. *IEEE Transactions on Visualization and Computer Graphics 11*, 2 (2005), 113–125.

[LvWJH04] LARAMEE R. S., VAN WIJK J. J., JOBARD B., HAUSER H.: Isa and ibfvs: Image space-based visualization of flow on surfaces. *IEEE Transactions on Visualization and Computer Graphics 10*, 6 (2004), 637–648.

[SH95] STALLING D., HEGE H.-C.: Fast and resolution independent line integral convolution. In *Proceedings of SIGGRAPH '95* (1995), ACM SIGGRAPH, pp. 249–256.

[SK97] SHEN H.-W., KAO D. L.: Uflic: a line integral convolution algorithm for visualizing unsteady flows. In *Proceedings of the conference on Visualization '97* (1997), ACM Press, pp. 317–323.

[SK98] SHEN H.-W., KAO D.: A new line integral convolution algorithm for visualizing time-varying flow fields. *IEEE Transactions on Visualization and Computer Graphics 4*, 2 (1998).

[vW02] VAN WIJK J.: Image based flow visualization. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), ACM Press, pp. 745–754.

[WE04] WEISKOPF D., ERTL T.: A hybrid physical/device-space approach for spatio-temporally coherent interactive texture advection on curved surfaces. In *Proceedings of the 2004 conference on Graphics interface* (School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004), Canadian Human-Computer Communications Society, pp. 263–270.

[WEE03] WEISKOPF D., ERLEBACHER G., ERTL T.: A texture-based framework for spacetime-coherent visualization of time-dependent vector fields. In *Proceedings of the 14th IEEE Visualization 2003* (Washington, DC, USA, 2003), IEEE Computer Society, p. 15.

[Wei04] WEISKOPF D.: Dye advection without the blur: A level-set approach for texture-based visualization of unsteady flow. *Computer Graphics Forum 23*, 3 (2004), 479–488.

[WSEE05] WEISKOPF D., SCHRAMM F., ERLEBACHER G., ERTL T.: Particle and texture-based spatiotemporal visualization of time-dependent vector fields. In *Proceedings of IEEE Visualization 2005* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 639 – 646.