Parallel Unsteady Flow Line Integral Convolution for High-Performance Dense Visualization

Zi'ang Ding*
Department of Computer Science,
Purdue University

Zhanping Liu†
Division of Computer Science,
Kentucky State University

Yang Yu‡
State Key Lab of CAD&CG,
Zhejiang University

Wei Chen§
State Key Lab of CAD&CG,
Zhejiang University

ABSTRACT

This paper presents an accurate parallel implementation of unsteady flow line integral convolution (UFLIC) for high-performance visualization of large time-varying flows. Our approach differs from previous implementations by using a novel value scattering+gathering mechanism to parallelize UFLIC and designing a pathline reuse strategy to reduce the computational cost of pathline integration. By exploiting the massive parallelism of modern graphical processing units (GPU), the proposed method allows for real-time dense visualization of unsteady flows with high spatial-temporal coherence.

Index Terms: I.3.3 [Computer Graphics]: Pictures / Image Generation

1 INTRODUCTION

Flow visualization plays an important role in processing, displaying, analyzing, and interpreting vector fields resulting from a wide variety of disciplines such as computational fluid dynamics simulation, oceanographic-atmospheric phenomena modeling, and electro-magnetic field analysis. There have been many geometry-based techniques[8] for visualizing flows ranging from steady to unsteady, and from planar to s urface and further to volume. Accelerated with graphics libraries and/or hardware, a lot of these methods enable real-time visualization. However, inappropriate seed distribution and insufficient control over the density of geometric primitives tend to incur an incomplete view or a cluttered image.

Texture-based flow visualization techniques [8] avoid the aforementioned problem by taking image space as a large seed set, usually one seed per pixel, to synthesize a dense depiction of the vector field. Instead of explicitly rendering intermediate graphical primitives (points, lines, and polygons) at discrete sample points or along seed-dependent integral traces, the dense representation created by texture-based techniques is a powerful way of conveying both local features and global patterns of a flow. In particular, unsteady flow line integral convolution (UFLIC) proposed by Shen and Kao [15, 16] provides a novel solution for visualizing time-varying flows with high image and animation quality. The use of a time-accurate value-scattering scheme coupled with a successive texture feed-forward strategy in UFLIC mimics the behavior of real-world flows to achieve high spatial-temporal coherence. However, the high computational cost due to a huge amount of pathline integration remains a major issue [12, 11, 10].

In this paper, we propose an accurate parallel implementation of UFLIC for high-performance dense visualization of time-varying flows. This method is unique for incorporating GPU-based pathline integration and texture synthesis with pathline reuse. Our approach runs at real-time frame rates while maintaining high spatial-temporal coherence.

The remainder of this paper is organized as follows. Section 2 introduces previous work closely related to our approach. Section 3 presents our parallel UFLIC algorithm, followed by a GPU implementation. In section 4, we provide some results to demonstrate both the efficiency and accuracy of the proposed method. We conclude this paper with a brief summary and outlook on future works in section 5.

2 RELATED WORK

The past two decades have seen significant research on texture-based techniques, as indicated in a comprehensive survey [8], while our discussion herein is primarily focused on those closely related to our work. Among the texture-based family, spot noise [18] and line integral convolution (LIC) [2] draw the most attention and have far-reaching impact. In particular, LIC has been widely used because of the high-resolution realistic representation and ease of implementation. The basic idea of LIC consists in the application of a 1.5D an-isotropic low-pass filter, shifted along the streamline that is symmetrically advected from each pixel center, to white noise to perform texture convolution. This image synthesis procedure emulates what occurs when a rectangular area of (massless) fine sand is blown by strong wind, exploiting and revealing the spatial correlation in terms of intensity that exists between the pixels hit by the same streamline. LIC is well suited for visualizing steady flows and there have been many variations, optimizations, and extensions such as oriented LIC [20], enhanced LIC [13], fast LIC [17], parallel LIC [22], and volume LIC [6, 14], to name only a few. However, it cannot be directly used to handle time-varying flows due to the inability to address temporal issues. Forssell and Cohen [3] proposed to use pathlines in place of streamlines to convey temporal coherence. Unfortunately, collecting texture values from downstream pixels does not comply with the real-world observation of the behavior of a time-varying flow and therefore their attempt fails to construct temporal coherence.

In contrast with LIC that employs an image-space oriented Eulerian-based value-gathering scheme, unsteady flow line integral convolution (UFLIC) adopts an object-space oriented Langrangian-based value-scattering mechanism to achieve high spatial coherence as well as strong temporal coherence. Specifically, scattering the footprint (or property/texture value) of a particle to its downstream locations along a pathline over several time steps not only correlates a considerable number of intra-frame pixels to create crispy images with accentuated flow streaks, but also correlates sufficient inter-frame pixels to produce a smooth animation. In addition, UFLIC uses a texture feed-forward strategy by which each output frame, after noise-jittered high-pass filtering, is taken as the input texture for synthesizing the next frame. In this way, an even tighter correlation can be established between any two consecutive frames to enhance temporal coherence. Compared to texture blending techniques such as LEA [7], IBFV [19], UFAC [21], and ISA [9] that incorporate

*e-mail:ding29@purdue.edu
†e-mail:zhanpingliu@hotmail.com
‡e-mail:yuyang@cad.zju.edu.cn
§e-mail:chenwei@cad.zju.edu.cn

IEEE Pacific Visualization Symposium 2015
14–17 April, Hangzhou, China
978-1-4673-6878-0/15/$31.00 ©2015 IEEE

single-step flow line advection with an exponentially-decaying low-pass filter, UFLIC demonstrates its advantage in maintaining high spatial-temporal coherence. One drawback is that the large computational cost restricts its applicability to interactive visualization of large flows.

The low computational performance of UFLIC is primarily due to the intensive process of advecting long flow lines that typically needs over 100 steps of integration for each pathline. In fact, there is a huge amount of redundancy in pathline integration between different particles during the value scattering process. In other words, different particles may leave their footprints or scatter their property values to a large number of common downstream pixels. Motivated by this observation, Liu and Moorhead proposed an accelerated UFLIC (AUFLIC) algorithm [11, 12] that runs one order-of-magnitude faster than UFLIC with the same image and animation quality. The essence of AUFLIC is to reuse pathlines both within and between value scattering processes, wherever possible and whenever possible, respectively. The ultimate goal is that in any value scattering process, one and only one seed (particle) is made available within each pixel, either explicitly placed from scratch at the center at the exact beginning of an integer time step or implicitly extracted from an existing pathline at an arbitrary position and/or at a fractional time, to move downstream as it scatters its texture value to its receiver pixels.

Li et al [10] presented a GPU-based implementation of UFLIC (GPUFLIC) that exploits the programmability of commodity GPUs to achieve interactive dense visualization of unsteady flows. To deal with the restriction that the single-instruction-multiple-data (SIMD) design of GPU imposes on data structures, they proposed an on-the-fly depositing scheme to obviate the need for a ring-bucket value accumulator as adopted in UFLIC. The interval between any two consecutive time steps is sub-divided into multiple time slots, of which each invokes a simplified, single-step value scattering process to generate a sub-frame. A sub-frame is obtained by performing convolution on the texture values that the upstream particles scatter within the current time slot. On an SIMD computing architecture, this time sub-division strategy effectively lengthens pathlines and accordingly increases the actual window of a convolution kernel. As the output animation is created with only the sub-frames corresponding to the original integer time steps (instead of the shorter time slots), GPUFLIC essentially emulates multi-step pathline integration and hence maintains higher spatial-temporal coherence than by the aforementioned single-step texture advection techniques [7, 19, 21, 9].

3 Our Method

Sharing the same idea of pathline reusing [11, 12] as well as GPU-based implementation to parallelize the pathline generation and value scattering [10], here we present our highly parallel unsteady flow line integral convolution algorithm. In our approach, the entire pipeline, particle management, value scattering and depositing, post-processing is fully CUDA-accelerated and achieves real time frame rates for each test dataset. Figure 1 shows the overview of our pipeline, and each step of our implementation will be discussed in this section.

3.1 Particle Management

At the beginning of each frame f, new particles are released at the center of every pixel in a domain of size $M \times N$, where M and N are the spatial resolution of the output image. Also an image value $\alpha = Tex^{f-1}$, where Tex^{f-1} is the image generated by the pervious frame $f-1$ (Tex^0 is the input white noise texture), is associated to each particle. This image value α remains unchanged during the entire life span of the particle.

During any frame the system has to maintain up to $K = M \times N \times L$ active particles, where L is the global particle life span measured in number of frames. After reaching the steady state, at each frame there are $M \times N$ particles reach the life span, hence the memory is recycled to be used by the new released particles.

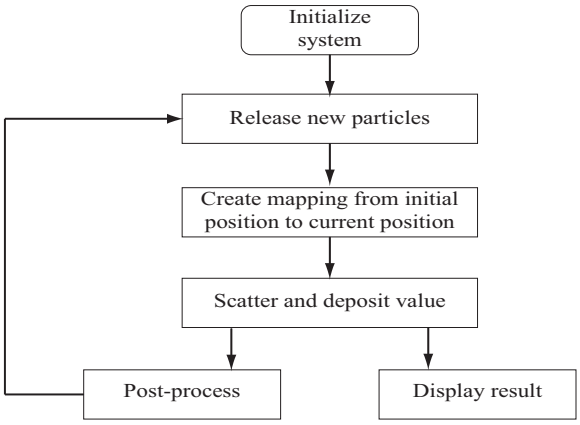

Figure 1: The pipeline of our algorithm.

Similar to the "on-the-fly depositing" method used in GPUFLIC, at every frame we only advect particles from their previous position to the current time instead of tracing them through their entire life span. The segment of a pathline in this time interval, named pathlet, is used to scatter the image value α stored in particle to pixels in current frame.

3.2 Pathline Reuse

After new particles released, all active particles in the system need to be advected forwardly by the unsteady flow. However the advection can be accelerated by parallelizing the computation on GPU [10], the redundancy in pathline integration [1] can be eliminated by a pathline reuse strategy [11, 12].

The strategy is based on an assumption that a pathlet traced by particle i at frame f can be reused by any particle j as long as i and j are in the same pixel at the beginning of this frame. Though those particles may diverge later, reusing pathlet has a very limited influence on the final result which will be discussed at section 4.

Figure 2 illustrates the pathline reuse strategy. Two pathlines Θ(red) and Φ(green) traced by particle θ and ϕ respectively, come into the same pixel at the beginning of frame f, the dot curves indicate the pathlets by continuing tracing θ and ϕ in this frame. Based on the above assumption each dot curve can be reused by the other. Furthermore, the system will release a new particle ψ(yellow) at the center of that pixel. Its pathlet Ψ can also be reused by θ and ϕ. Thus, at each frame, only the new released $M \times N$ particles need to be advected to generate the new pathlets. All other active particles reuse those pathlets during the value scattering and gathering step.

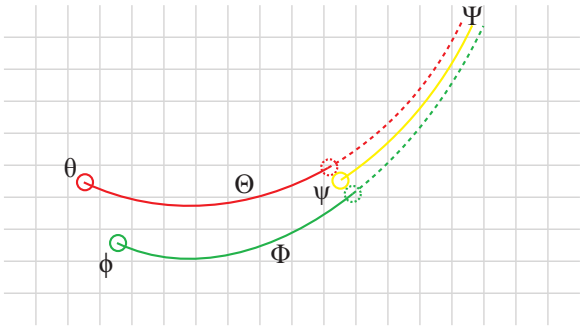

Figure 2: Illustration of our pathline reuse strategy.

3.3 Pathline Reuse based Value Scattering and Depositing

At every frame, the image value α associated with each particle needs to be scattered over all pixels covered by its pathlet. With our pathline reuse strategy, all particles at the same pixel will share a newly created pathlet and therefore their associated image values can be scattered together by the same pathlet. In contrast, GPUFLIC needs to scatter the value separately because all particles are advected individually.

When the system releases new particles, each of them is stored based on the initial starting position. In order to perform the value scattering and depositing by using the benefit of the pathline reuse strategy, each particle has to be fetched based on its current position. This can be achieved by creating a mapping from particle's current position to its initial position.

The mapping process contains following three operations:

1. **Pixel Particle Counting** Each pixel counts the number of particles that located in it at the starting time of current frame.
2. **Pixel Offset Computing** A prefix sum operation applied on the result of previous operation, and generates the offset for each pixel in the mapping buffer.
3. **Particle to Pixel Mapping** Each particle maps itself to the pixel by the current position and stores it to the mapping buffer based on the offset computed in the previous operation.

Figure 3 illustrates the result after the mapping process. Each pixel has a unique offset which point to the memory location in the mapping buffer and the number of particles inside it, while the mapping buffer contains the actual memory address of each particle.

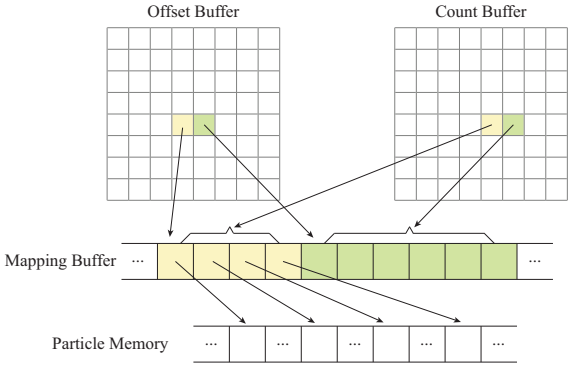

Figure 3: Illustration of the result after the mapping process.

The process of value scattering and depositing begins with tracing a seed $s(x,y)$ placed in the center of each pixel $p(x,y)$. In each step, a seed $s(x,y)$ is advected by solving an ordinary differential equation on the input unsteady vector field. We employ higher-order numerical integration methods such as 4th order Runge-Kutta to compute the new position POS_k from the previous one POS_{k-1}. All particles located at the pixel $p(x,y)$ corresponding to that seed $s(x,y)$, which could be fetched by the mapping buffer, scatter their associated image values by depositing them to the pixels on the image plane covered by the line segment $\overline{POS_{k-1}POS_k}$. We use one buffer $\mathscr{B}_V$ to store the sum of weighted values $\sum V \times w$, and another one $\mathscr{B}_w$ to store the accumulated weights $\sum w$. In our implementation, each weight is determined by two factors, namely, the life time of the particle and the distance of the particle travelled in a pixel.

After scattering and depositing all values to the image plane, each particle has to update its position and a scalar value τ, denoting remaining life time, which is initialized as the global life span L. Then the weighted average is computed by taking the division of those two buffers $\frac{\sum V \times w}{\sum w}$. This weighted average can be displayed as a grayscale image result of current frame, moreover, a color coding based on the magnitude of local velocity, some property scalar fields, or the FTLE(*Finite-Time Lyapunov Exponent*) [5] field can be used to generate a color image result.

3.4 Post-Processing

Mentioned in UFLIC [15, 16], the LIC method is really a low-pass filtering process on the input image. Simply using the current result as the input of next frame would lead to an over-blurred image. Postprocessing is required to maintain a satisfying contrast of the result throughout the animation.

The first step of the postprocessing is applying a high-pass filter to enhance the image contrast. Similar to the UFLIC, the following Laplacian operator is applied to the current result:

$$\begin{vmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{vmatrix} = \begin{vmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{vmatrix} - \begin{vmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{vmatrix} \quad (1)$$

In order to prevent the unnecessary high frequencies introduced by the high pass filter, we apply a second step, named noise jittering, to the output from the high-pass filter. Similar to the GPUFLIC, to address the frozen patterns in steady regions, this step replaces the lower seven bits of the output by a periodic noise.

3.5 CUDA Implementation

The above presented algorithm can be fully parallelized on the massive parallel architecture such as CUDA (Compute Unified Device Architecture)[10]. Although most parts of the CUDA-based codes are identical to C code on CPU, there are still some details need to be presented in our implementation.

3.5.1 Memory

As mentioned before, at any frame the number of total particles in the system is bounded by $K = M \times N \times L$. Thus, a pre-allocated memory on the global memory in CUDA is used to store the information of particles, namely, the current position *POS*, the image value α, and the remaining life time τ. This memory is divided into L layers, each layer is further divided into $M \times N$ blocks which store the information of one particle. At any frame f, all particles in the layer $f\%L$ (f mod L) will reach their life span, and reuse by the new released particles. Please refer to Figure 4 as the layout of the global memory for particles.

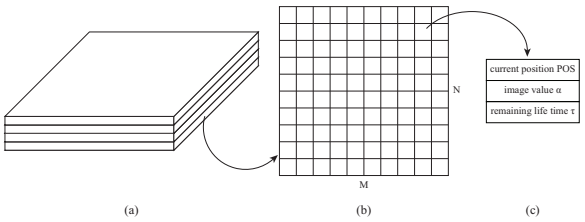

Figure 4: The layout of the global memory for particles.(a) the whole memory is divided into L layers(4 in this figure); (b) each layer is divided into $M \times N$ blocks; (c) each block contains the information of one particle.

The input 2D unsteady vector field is stored as a 3D texture memory in CUDA, which is cached and no extra cost on trilinear interpolation. Each slice on the z axis corresponds to a time step in the 2D unsteady vector field. With this configuration, the spatial-temporal coordinate (x,y,t), where t is the global time, is transformed to the texture coordinate (u,v,w) in our implementation to access the flow velocity at any position of any time.

3.5.2 Concurrency

A race condition arises when two or more threads in CUDA attempt to access the same global memory concurrently and at least one access is a write operator. In our algorithm the value scattering and depositing step will trigger the race condition, for example, two threads attempt to update the value stored in the same memory address of $\mathscr{B}_V$ at the same time. Atomic operators supported by CUDA such as, **atomicAdd**, are used in order to avoid unexpected results caused by the race condition.

3.5.3 Bandwidth

To maximally optimize the use of the bus bandwidth, our algorithm is implemented entirely on CUDA. At the initial state of the program, the input unsteady vector field and white noise image are transferred from the host memory to the video memory. After that, no data transfer is needed. The result of each frame is displayed effectively by using the pixel buffer object feature supported by CUDA.

4 Result and Discussion

We implemented our algorithm described in section 3 using C++ and CUDA on a Windows PC. The platform we used in our experiment has a hex-core i7 processor at 3.2GHz with 12GB of RAM and an NVIDIA GTX590 CUDA-Enabled graphics card (dual GPUs, 1536MB video RAM per GPU).

Four time-varying vector fields, including PSI(figure 5), double gyre(figure 6), wind flow(figure 7), and convection(figure 8) were tested in our experiment. The life span used in each test case is listed in table 1. Both PSI and Double Gyre are standard synthetic datasets, while wind flow and convection are results from the numerical simulation of certain physical phenomena. The color coding in figure 5 and figure 7 is based on the magnitude of local velocity, while in figure 8 it highlights the temperature. FTLE color coding is used in figure 6. For more information about the FTLE color coding, please refer to [4].

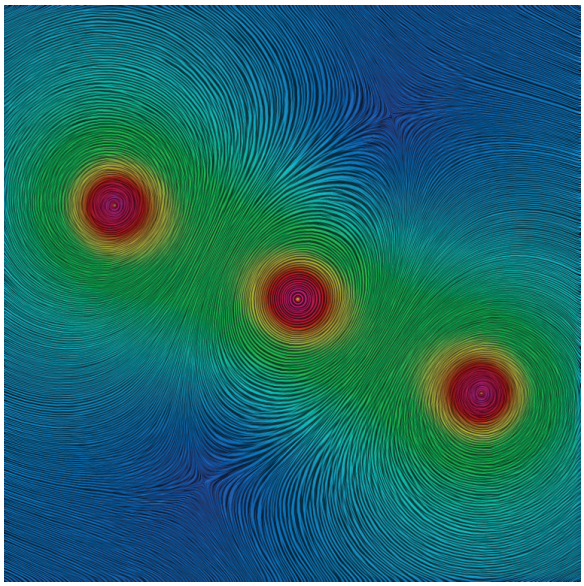

Figure 5: PSI dataset.

Table 1 gives the performance of each dataset by our implementation. In all cases we are able to achieve real-time frame rates at different image resolutions. We have implemented GPUFLIC algorithm and make the performance comparison in the same hardware platform. Table 1 indicates our algorithm outperforms it because of the pathline reuse strategy. It is apparent in figure 9 that the performance of GPUFLIC decreases faster and runs less stable than our approach with the increasing of the life span.

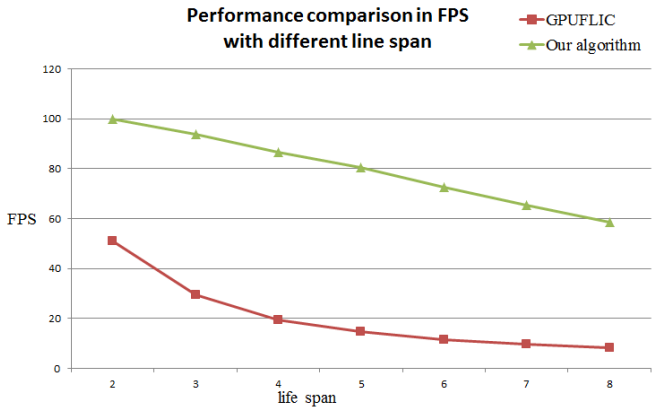

Figure 9: Performance comparison in FPS with different life spans. All results are tested on PSI dataset with resolution of 512×512.

In order to evaluate the influence of pathline reuse, we made a side by side comparison between our approach and GPUFIC. The first row in figure 10 shows the result of our approach on PSI dataset without color coding, and the second row is the result of GPUFLIC at corresponding frames. Our approach is able to capture the same high quality crisp line patterns matching those generated by GPUFLIC.

5 Conclusion and Future Work

We have presented a GPU-based parallel algorithm that synthesizes LIC-like textures at real-time frame rates for dense visualization of time-varying 2D flows. It is built on UFLIC to not only convey very high spatial coherence in individual frames but also exhibit very strong temporal coherence in the resulting animation. Our algorithm runs faster than AUFLIC by taking advantage of GPU parallelism while it differs from GPUFLIC due to a pathline reuse mechanism as well as a relatively efficient on-the-fly value scattering strategy. The results show that the proposed approach has a great potential, in both visualization quality and computational performance, to bring UFLIC to practical applications.

As for future work, we would like to extend our algorithm for curved surface and volumetric flows. We are also interested in exploring the use of an adaptive step size integration method in pathline advection.

Acknowledgements

The authors wish to thank Prof. Xavier Tricoche of Purdue University for his helpful suggestions and comments. This work was supported by Major Program of National Natural Science Foundation of China (61232012) and National Natural Science Foundation of China (61422211).

References

[1] S. L. Brunton and C. W. Rowley. Fast computation of finite-time lyapunov exponent fields for unsteady flows. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 20(1):–, 2010.

[2] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Proceedings of ACM SIGGRAPH*, pages 263–270, 1993.

[3] L. K. Forssell and S. D. Cohen. Using line integral convolution for flow visualization: Curvilinear grids, variable-speed animation, and

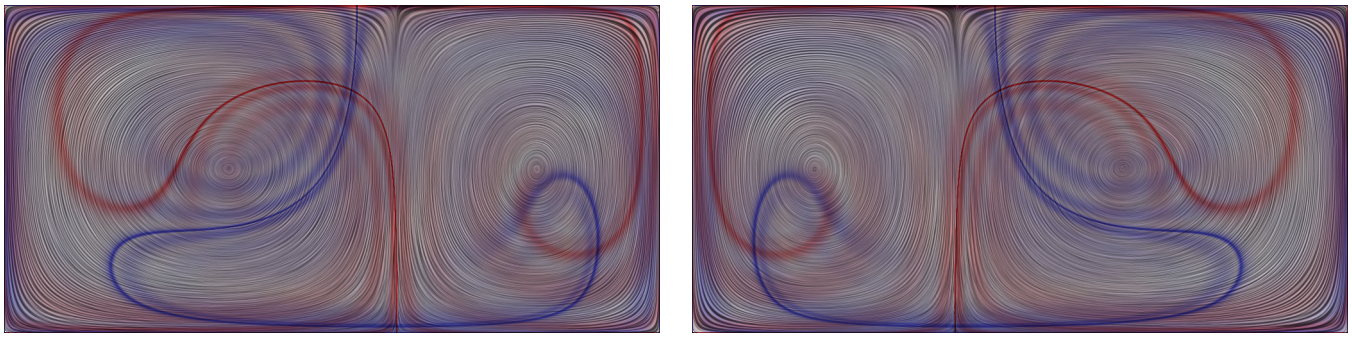

Figure 6: Double gyre dataset.

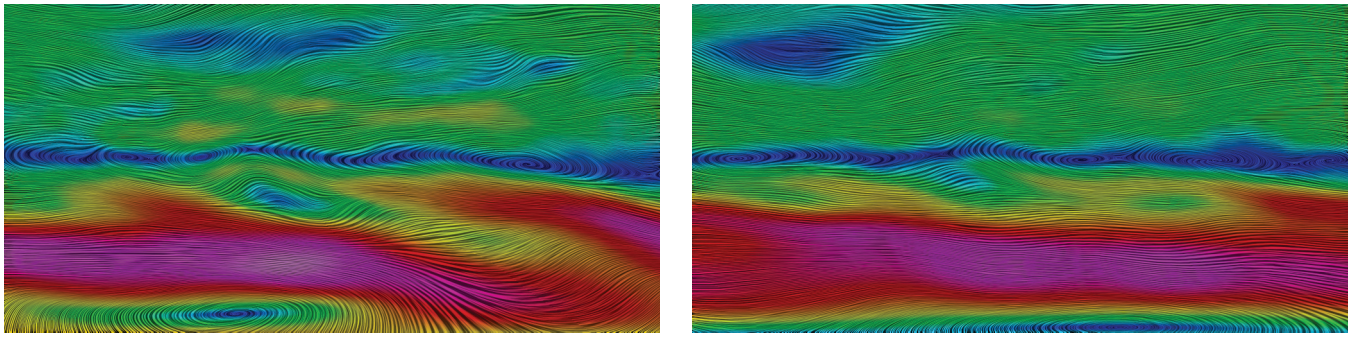

Figure 7: Wind flow dataset.

unsteady flows. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):133–141, 1995.

[4] C. Garth, G.-S. Li, X. Tricoche, C. Hansen, and H. Hagen. Visualization of coherent structures in transient 2d flows. In H.-C. Hege, K. Polthier, and G. Scheuermann, editors, *Topology-Based Methods in Visualization II*, Mathematics and Visualization, pages 1–13. Springer Berlin Heidelberg, 2009.

[5] G. Haller. Distinguished material surfaces and coherent structures in three-dimensional fluid flows. *Physica D: Nonlinear Phenomena*, 149(4):248 – 277, 2001.

[6] V. Interrante and C. Grosch. Strategies for effectively visualizing 3D flow with volume LIC. In *Proceedings of IEEE Visualization*, pages 421–ff., 1997.

[7] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Lagrangian-eulerian advection for unsteady flow visualization. In *Proceedings of IEEE Visualization*, pages 53–60, 2001.

[8] R. S. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F. H. Post, and D. Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum*, 23:2004, 2004.

[9] R. S. Laramee, J. J. van Wijk, B. Jobard, and H. Hauser. ISA and IBFVS: Image space-based visualization of flow on surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 10(6):637–648, 2004.

[10] G.-S. Li, X. Tricoche, and C. Hansen. GPUFLIC: Interactive and accurate dense visualization of unsteady flows. In *Proceedings of Eurographics/IEEE VGTC Symposium on Visualization*, pages 29–34, Lisbon, Portugal, 2006.

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

[12] Z. Liu and I. Robert James Moorhead. AUFLIC: an accelerated algorithm for unsteady flow line integral convolution. In *Proceedings of the Symposium on Data Visualisation*, pages 43–51, 2002.

[13] A. Okada and D. Lane. Enhanced line integral convolution with flow feature detection. In *Proceedings of SPIE Visual Data Exploration and Analysis IV*, pages 206–217, 1997.

[14] C. Rezk-Salama, P. Hastreiter, C. Teitzel, and T. Ertl. Interactive exploration of volume line integral convolution based on 3D texture mapping. In *Proceedings of IEEE Visualization*, pages 233–240, 1999.

[15] H.-W. Shen and D. L. Kao. UFLIC: a line integral convolution algorithm for visualizing unsteady flows. In *Proceedings of IEEE Visualization*, pages 317–325, 1997.

[16] H.-W. Shen and D. L. Kao. A new line integral convolution algorithm for visualizing time-varying flow fields. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):98–108, 1998.

[17] D. Stalling and H.-C. Hege. Fast and resolution independent line integral convolution. In *Proceedings of ACM SIGGRAPH*, pages 249–256, 1995.

[18] J. J. van Wijk. Spot noise texture synthesis for data visualization. In *Proceedings of ACM SIGGRAPH*.

[19] J. J. van Wijk. Image based flow visualization. *ACM Transactions on Graphics*, 21(3):745–754, 2002.

[20] R. Wegenkittl, E. Groller, and W. Purgathofer. Animating flow fields: Rendering of oriented line integral convolution. In *Proceedings of IEEE Computer Animation*, pages 15–23, 1997.

[21] D. Weiskopf, G. Erlebacher, and T. Ertl. A texture-based framework for spacetime-coherent visualization of time-dependent vector fields. In *Proceedings of IEEE Visualization*, pages 15–23, Washington DC, USA, 2003.

[22] M. Zöckler, D. Stalling, and H.-C. Hege. Parallel line integral convolution. *Parallel Computing*, 23(7):975–989, 1997.

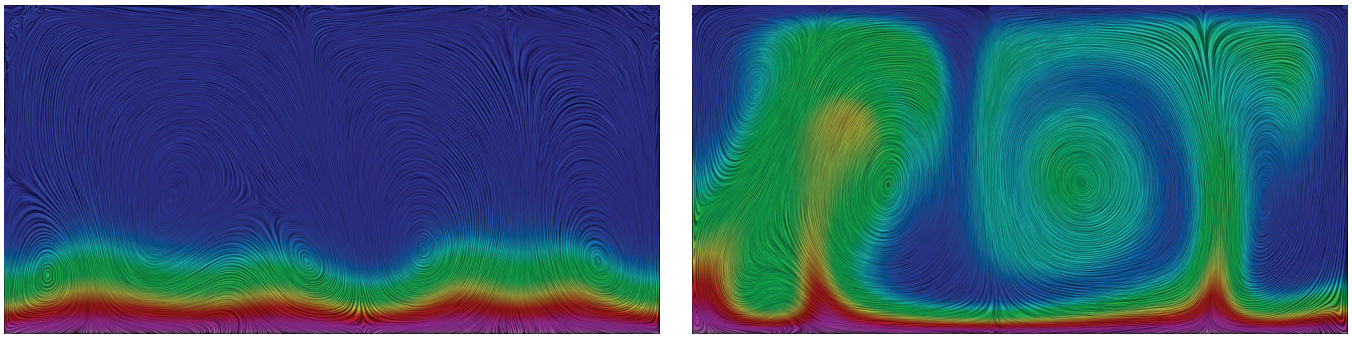

Figure 8: Convection dataset.

Table 1: Performance in average FPS.

Dataset	Resolution	Time steps	Life span	GPUFLIC (FPS)	Ours (FPS)	Speedup Ratio
PSI	512×512	101	4	19.5	86.6	4.4
PSI	1024×1024	101	4	7.0	31.9	4.6
Double Gyre	1024×512	100	4	14.1	59.3	4.2
Wind Flow	1024×512	41	4	17.4	65.4	3.7
Convection	1024×512	200	4	11.5	48.2	4.2

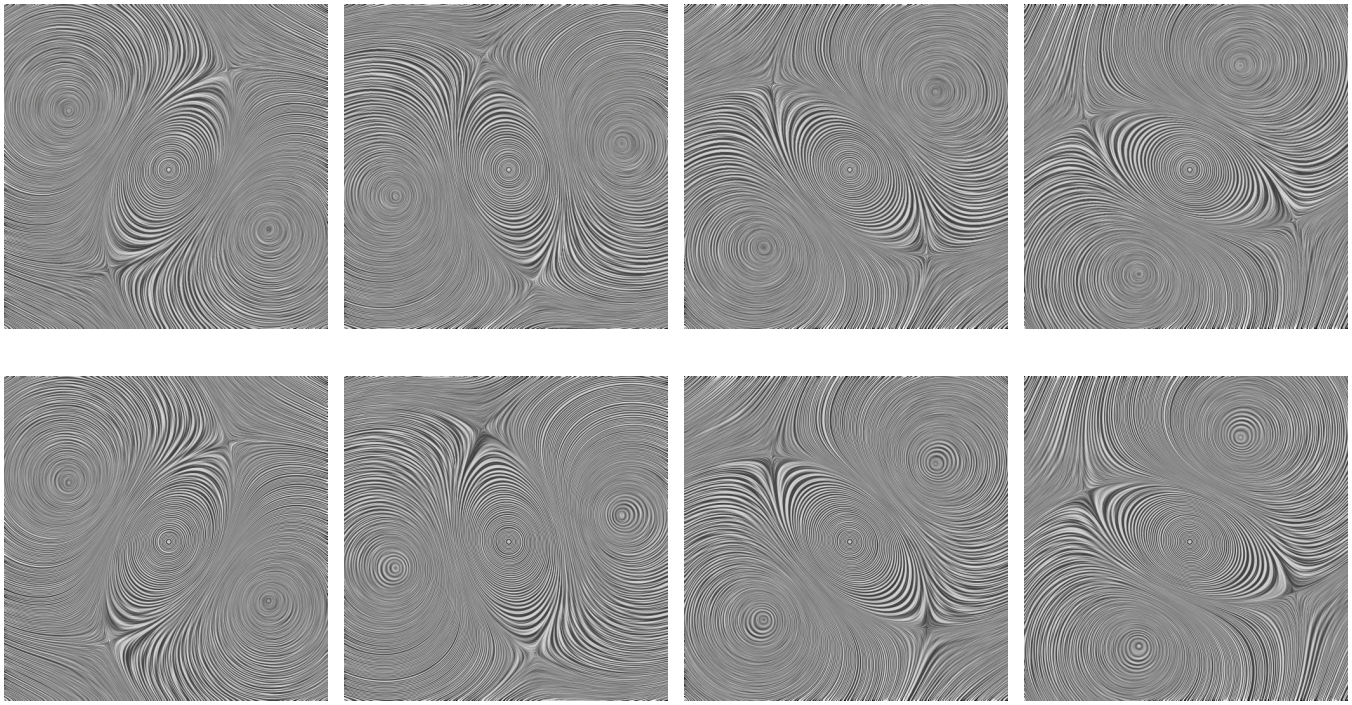

Figure 10: A side-by-side comparison between our approach and GPUFLIC on PSI dataset.