Eurographics Symposium on Parallel Graphics and Visualization (2018)
H. Childs, F. Cucchietti (Editors)

Dense Texture Flow Visualization using Data-Parallel Primitives

M. Kim[1] and S. Klasky[1] and D. Pugmire[1]

[1]Oak Ridge National Laboratory, Oak Ridge TN., USA

Abstract

Recent trends in supercomputing towards massively threaded on-node processors to increase performance has also introduced fragmented software support. In response to this changing landscape, new scientific visualization packages have been developed to provide a portable framework to exploit this on-node parallelism with data parallel primitives, while also providing a single interface to multiple hardware backends. This necessitates adapting algorithms to the data parallel primitives paradigm. In numerous cases the algorithm is serial, but other times the technique is tied to hardware and needs to be generalized to broadly disseminate.

In this work, we present unsteady flow line integral convolution (UFLIC) using only data parallel primitives. Line integral convolution (LIC) is a fundamental flow visualization technique in scientific visualization. LIC and its texture-based variants, are used in fields such as meteorology and computational fluid dynamics to aid practitioners because of its efficient memory usage, strong, visual flow characteristics, and efficient performance. However, in practice performant implementations are GPU shader-based approaches, which limits deployment and adoption. By utilizing VTK-m, our approach is a performant, memory efficient implementation, with the added benefit of portability, with a single implementation across many architectures.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

The modern HPC world has induced a change in scientific visualization and analysis. Accelerators such as Nvidia Tesla have lead the way in increasing compute through massive on-node parallelization for simulation codes. This increasingly heterogenous architecture landscape has lead to a fragmented software ecosystem, with various software infrastructure such as CUDA [NVI07] or Thread Building Blocks [Rei]. Unfortunately, developers are wary of porting algorithms to various frameworks because of the time needed to refactor and optimize per architecture.

Until recently, the state-of-the-art software frameworks did not have support for this parallelism across the different types of processors. For visualization, the multicore visualization toolkit, VTK-m, was introduced, using data parallel primitives (DPP). DPP is a paradigm that applies a set of functions, scan, map, reduce, etc. to vectors of data [Ble90]. This allows for a write-once run-anywhere approach where DPP, in conjunction with a visualization data model, can be run on a myriad of parallel processors.

To exploit DPP in VTK-m, traditional techniques need to be adapted. One such technique is line integral convolution (LIC), which is a fundamental flow visualization technique in scientific visualization [CL93]. LIC, and its unsteady variants [SK97, SK98], are used in various fields such as meteorology and computational fluid dynamics because of its efficient memory usage, strong, visual characteristics, and performance. Many of the accelerated approaches were implemented with GPU shaders, which explicitly ties it to the hardware and are not generally runnable on other hardware.

In this paper, we introduce a data-parallel-primitives-based unsteady flow line integral convolution (UFLIC) using VTK-m. By using VTK-m, the UFLIC is portable across multiple software infrastructures while maintaining strong parallel performance. The rest of this work is as follows: Sec. 2 is a review of the previous works. A description of DPP is given in Sec. 3 and in Sec. 4 UFLIC using DPP is stated. Finally, Sec. 5 is a discussion of performance results on various platforms and Sec. 6 contains our conclusions.

2. Previous Works

In this section, a high level overview of parallel visualization is in Sec. 2.1 and hardware accelerated unsteady flow visualization is reviewed in Sec. 2.2.

2.1. Parallel Visualization

Historically, many visualization techniques were implemented in a serial fashion, and with the adoption of manycore/multicore accelerators in HPC, a new paradigm is required to extract performance. These manycore/multicore accelerators have potentially thousands

DOI: 10.2312/pgv.20181095

of concurrent threads and have new APIs, such as CUDA [NVI07], OpenCL [SGS10], and Thread Building Blocks (TBB) [Rei] that enables that massive parallelism. Thrust [HB09] is a proprietary parallel primitives [Ble90] API for programming on Nvidia hardware based on CUDA. Along with parallel primitives, such as scan or map, the API allows for "functors" to be applied to data in a parallel manner.

To adapt to the heterogeneous, manycore/multicore landscape, several visualization packages were developed: PISTON [LSA12], Dax [MKMM12], and EAVL [MAPS12]. PISTON is built on CUDA, and has a general parallel programming model. Dax hews more closely to the data problems in scientific visualization and creates parallelization over each element of a mesh. Finally, EAVL tackles the problems of the modern visualization data model and can handle increasing complex data models in a parallel fashion. Recently, the three packages were merged into VTK-m, a multicore, data parallel primitive toolkit for scientific visualization [MSU*16].

2.2. Unsteady Line Integral Convolution

Flow visualization is an expansive topic, so it is limited here to texture-based, LIC variants and hardware accelerated approaches. For a more thorough overview, we suggest reading [LHD*04].

Line integral convolution (LIC), was introduced by Cabral and Leedom [CL93]. To perform the LIC, for each pixel, bi-directional streamlines are generated to gather the intensity values along the streamline and a low-pass filter is applied to the resulting accumulated values. Unfortunately, this gather is slow, which [SH95] addresses to speed-up LIC. Forssell and Cohen [FC95] adapted LIC for curvilinear grid data, but has poor spatial patterns for unsteady flow. Shen and Kao [SK97, SK98] focused on unsteady flow LIC (UFLIC). Instead of gathering values for each pixel, UFLIC scatter values along pathlines. Further, a circular queue is deployed instead of a linked-list for accumulating intensity values.

For hardware accelerated approaches, Jobard, Erlebacher, and Hussaini [JEH01] performs backwards pathline integration. This technique generates good temporal coherence, but is noisy spatially. Similarly, van Wijk [vW02] performs image advection and alpha-blending, which also has good temporal coherence, but is noisy. Finally, Li, Tricoche, and Hansen [LTH06] exploited the GPU for hardware acceleration for interactive, parallel UFLIC. This emulates a multi-step pathline integration over the UFLIC ring buffer [SK98]. Value depositing along pathlines is the most time consuming aspect of UFLIC because much of the work is to deposit values into the future. To increase the performance, instead of tracing pathlines throughout their entire life time, Li et al. [LTH06] use "pathlets," advecting particles from their current position to the next position. This way a ring-bucket per pixel is no longer required [SK98], rather only time-to-live (ttl) number of frames are computed in each iteration.

3. Data Parallel Primitives

Data parallel primitives [Ble90] is the abstraction model used by VTK-m to provide performant, portable code. VTK-m provides four data parallel primitives. The *map* operator applies the same operator to every element of a vector. The *reduce* operation computes a single value from all the values of a vector, such as the sum of the elements of a vector. The *scan* operation is similar to the reduce operation, except instead of a single value, multiple partial reductions are stored to an output vector. The *gather/scatter* operators perform a parallel copy. A gather operation brings many values into a single element of a vector. A scatter operation mirrors the gather operation, and distributes a single value into many different elements of a vector. By using these operators as the basis for parallel operations in VTK-m, it can target multiple architectural backends.

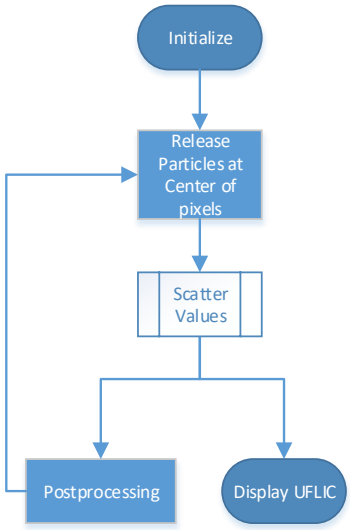


Figure 1: *An overview of the parallel primitives UFLIC.*

4. Parallel Primitives UFLIC

The parallel primitives UFLIC method is similar to [LTH06]. Fig. 1 is an overview of the algorithm. Initially, a particle is placed at the center of every pixel. Further, a white noise is used to seed the initial intensity values of the particles. The particles are advected and the intensity of the particle is deposited along the pathline of the particle.

Once all the particles have been advected, the accumulated intensities are normalized, sharpened and jittered for the next iteration. In the next iteration I, the previous particles continue from their current position, and new particles are placed at the center of every pixel with a new white noise, all the particles are advected, intensities deposited, normalized, sharpened, and jittered. This continues for T iterations, where T is the "time to live": this defines how long the particle will live by number of iterations. Once the number of iterations is larger than the time to live, the texture and particles are reused.

4.1. Particle Advection

For every iteration I, initially a particle, p_i is placed at the center of every pixel in the current texture of size $M \times N$. The particle is advected through the velocity field using a Runge-Kutta 4th order integration, to the new position p_i^{next}. This unsteady vector field integrator is a DPP *map* function, which applies the integrator to each particle in the data array. The path from p_i to p_i^{next} is the pathlet: a portion of a pathline for that iteration's time step.

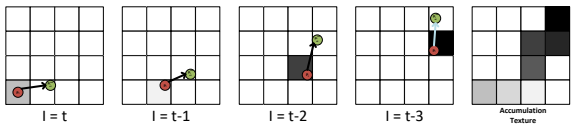


Figure 2: *An example of particles advected and depositing their values into the accumulation texture.*

For unsteady flow, there is a "time to live" (T) which defines how long the particle will live by number of iterations. Therefore, there are $M \times N \times T$ particles to compute the current iteration of the UFLIC. Particles are advected from their current position and continue to carry their initial α_i intensity value to scatter along the accumulation texture. Once a particle has surpassed its T, it is killed and no longer advected. In practice, the particles that are "killed" are recycled and used again as newly released particles. An example is in Fig. 2: the time to live is $T = 4$ and the three previous iterations, $t-3$, $t-2$, and $t-1$ all advect the particle from their current position to the next position.

4.2. Value Depositing

To generate the UFLIC, the intensity values are *scattered* into an accumulation texture. Initially, the intensity value α_i for a particle p_i is fetched from the current intensity texture, Tex_I, which is derived from the previous intensity texture, Tex_{I-1}. The initial texture, Tex_0 is filled with a white noise. For each iteration I, for each pathlet from p_i to p_i^{next}, *scatter* the intensity value α_i on the line of the pathlet with a line algorithm [Bre65]. Further, the number of deposits per pixel is stored in ω_i. For unsteady flow, all the pathlets ($M \times N \times T$) write to the same output texture, the accumulation texture. Once all depositing is completed, the accumulation texture is normalized by the ω number of deposits and the normalized values are stored in Tex_I.

4.3. Post-processing

The scatter process (Sec. 4.2) is diffusive, so the accumulation texture is sharpened with the Laplacian filter for the next iteration. Further, the accumulation texture is "jittered" to increase the contrast, in time. Finally, this sharpened and jittered texture is used as the intensity values of the next iteration. The sharpening and jittering filters are *map* operations, where a 2D stencil is applied to each data array value.

5. Results

This algorithm is implemented using C++ and VTK-m with an Intel Core i7-5960X and an Nvidia GTX 980 GPU. Three datasets were used to demonstrate the flexibility of the algorithm: a time-varying double gyre (Fig. 3), the PSI dataset (Fig. 4(b)), and an XGC fusion dataset [CKD*09] (Fig. 4(d)). The double gyre and PSI are standard synthetic datasets for two-dimensional flow visualization with the double gyre dataset used as a scaling test as well. The XGC dataset is a fusion tokamak simulation code which generates a twisting vector field. By visualizing the twisting field, scientists can quickly determine whether the simulation is correct.

The timing results are in Table 1 and speed-up results are in Table 2. VTK-m supports serial, TBB, and CUDA backends, and for the double gyre dataset the scaling goes from 512×256 to 4096×2048. All tests were done with a ttl of 4, and a total iteration count of 12. CUDA is up to $222\times$ faster over the serial implementation, which is a good speed-up result. Similarly, an up to $28\times$ speed-up of CUDA over multi-threaded TBB is reasonable increase in performance. Finally, PSI and XGC (with an image size of 512×512) had similar speed-ups when comparing CUDA and Serial ($58.3\times$ and $48.4\times$, respectively), within expected performance increase when comparing a multi-threaded CPU implementation to a GPU implementation.

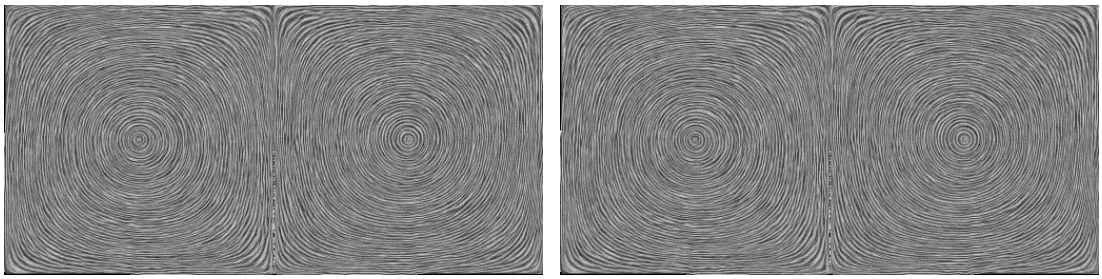
The original GPUFLIC was written in the Nvidia Cg shading language which is no longer under active development. Therefore, we implemented the UFLIC algorithm in CUDA (CUFLIC) to compare the visual accuracy (Fig. 3 and 4) and performance of a strictly CUDA/Thrust implementation against the VTK-m CUDA (VCUFLIC) implementation. Visually, the double gyre, PSI, and XGC LIC generated with VCUFLIC are strongly similar to the LIC generated with CUFLIC, retaining the smooth line patterns. For performance results, VCUFLIC is compared with CUFLIC in Table 2 because it is the fastest of the VTK-m backends. With the double gyre dataset, the speed-up of CUFLIC over VCUFLIC starts at $3.0\times$ with an image size of 512×256 but reduces to $1.3\times$ at a grid size of 4096×2048. This indicates that there is a overhead cost to invoking the VTK-m library. The overhead cost includes creation of the VTK-m worklet, which includes increased register count for VTK-m. However, this overhead is a static cost, i.e. it does not increase as the amount of work increase.

6. Conclusion

In this work we present an unsteady flow line integral convolution using only data parallel primitives. By using data parallel primitives, the UFLIC algorithm can be deployed to multiple architectures, enabling performant flow visualization while maintaining the high quality, smooth line patterns similar to previous solutions.

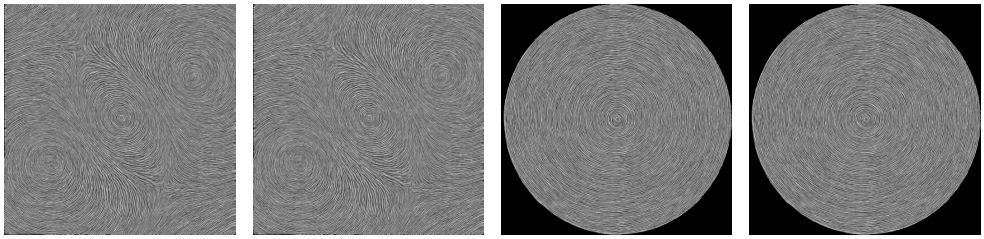
In the future we would like to expand this work to 2.5D surface and 3D flow visualization. Further, we would like to explore an adaptive time step integration.

The authors would like to thank the reviewers for their generous feedback. This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.



(a) CUDA Double Gyre (b) VTK-m Double Gyre

Figure 3: *Figures 3(a) and 3(b) are the double gyre (512×256) implemented with CUDA and VTK-m, respectively.*



(a) PSI using CUDA. (b) PSI using VTK-m. (c) XGC using CUDA. (d) XGC using VTK-m.

Figure 4: *Fig. 4(a) is the PSI dataset using a CUDA implementation and Fig. 4(b) uses VTK-m. Similarly, Fig. 4(c) is the XGC dataset using a CUDA implementation and Fig. 4(d) uses VTK-m.*

Table 1: *The timing results (in seconds) for the double gyre, PSI, and XGC datasets. The double gyre is scaled from 512×256 to 4096×2048. The results are for VTK-m with a serial, TBB, and CUDA backend (VCUFLIC), as well as a CUDA with Thrust (CUFLIC) implementation.*

Dataset	Dimensions	VTK-m			CUFLIC
		Serial	TBB	CUDA	
Double Gyre	512x256	2.131	0.412	0.033	0.011
	1024x512	8.622	1.225	0.065	0.033
	2048x1024	33.392	4.754	0.185	0.119
	4096x2048	132.839	16.676	0.596	0.452
PSI	512x512	2.320	0.467	0.040	0.025
XGC	512x512	1.896	0.433	0.039	0.016

Table 2: *The speed-up results for the double gyre, PSI, and XGC datasets. The double gyre is scaled from 512×256 to 8192×4096. The results are for VTK-m with a serial, TBB, and CUDA backend (VCUFLIC), as well as a CUDA with Thrust (CUFLIC) implementation.*

Dataset	Dimensions	VTK-m			
		Serial vs TBB	Serial vs CUDA	TBB vs CUDA	VCUFLIC vs CUFLIC
Double Gyre	512x256	5.175×	64.525×	12.468×	3.009×
	1024x512	7.038×	131.942×	18.747×	1.961×
	2048x1024	7.023×	180.892×	25.756×	1.553×
	4096x2048	7.966×	222.800×	27.969×	1.320×
PSI	512x512	4.969×	58.293×	11.730×	1.585×
XGC	512x512	4.381×	48.813×	11.141×	1.547×

References

[Ble90] BLELLOCH G. E.: *Vector Models for Data-parallel Computing*. MIT Press, Cambridge, MA, USA, 1990. 1, 2

[Bre65] BRESENHAM J. E.: Algorithm for computer control of a digital plotter. *IBM Syst. J. 4*, 1 (Mar. 1965), 25–30. URL: http://dx.doi.org/10.1147/sj.41.0025, doi:10.1147/sj.41.0025. 3

[CKD*09] CHANG C. S., KU S., DIAMOND P. H., LIN Z., PARKER S., HAHM T. S., SAMATOVA N.: Compressed ion temperature gradient turbulence in diverted tokamak edge. *Physics of Plasmas 16*, 5 (2009), 056108. URL: https://doi.org/10.1063/1.3099329, arXiv:https://doi.org/10.1063/1.3099329, doi:10.1063/1.3099329. 3

[CL93] CABRAL B., LEEDOM L. C.: Imaging vector fields using line integral convolution. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1993), SIGGRAPH '93, ACM, pp. 263–270. URL: http://doi.acm.org/10.1145/166117.166151, doi:10.1145/166117.166151. 1, 2

[FC95] FORSSELL L. K., COHEN S. D.: Using line integral convolution for flow visualization: curvilinear grids, variable-speed animation, and unsteady flows. *IEEE Transactions on Visualization and Computer Graphics 1*, 2 (Jun 1995), 133–141. doi:10.1109/2945.468406. 2

[HB09] HOBEROCK J., BELL N.: Thrust: A parallel template library. *Thrust: A Parallel Template Library* (2009). 2

[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), VIS '01, IEEE Computer Society, pp. 53–60. URL: http://dl.acm.org/citation.cfm?id=601671.601678. 2

[LHD*04] LARAMEE R. S., HAUSER H., DOLEISCH H., VROLIJK B., POST F. H., WEISKOPF D.: The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum 23*, 2 (2004), 203–221. URL: http://dx.doi.org/10.1111/j.1467-8659.2004.00753.x, doi:10.1111/j.1467-8659.2004.00753.x. 2

[LSA12] LO L.-T., SEWELL C., AHRENS J.: PISTON: A Portable Cross-Platform Framework for Data-Parallel Visualization Operators. In *Eurographics Symposium on Parallel Graphics and Visualization* (2012), Childs H., Kuhlen T., Marton F., (Eds.), The Eurographics Association. doi:10.2312/EGPGV/EGPGV12/011-020. 2

[LTH06] LI G.-S., TRICOCHE X., HANSEN C.: GPUFLIC: Interactive and Accurate Dense Visualization of Unsteady Flows. In *EUROVIS - Eurographics /IEEE VGTC Symposium on Visualization* (2006), Santos B. S., Ertl T., Joy K., (Eds.), The Eurographics Association. doi:10.2312/VisSym/EuroVis06/029-034. 2

[MAPS12] MEREDITH J. S., AHERN S., PUGMIRE D., SISNEROS R.: EAVL: The Extreme-scale Analysis and Visualization Library. In *Eurographics Symposium on Parallel Graphics and Visualization* (2012), Childs H., Kuhlen T., Marton F., (Eds.), The Eurographics Association. doi:10.2312/EGPGV/EGPGV12/021-030. 2

[MKMM12] MORELAND K., KING B., MAYNARD R., MA K.-L.: Flexible analysis software for emerging architectures. In *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis* (Washington, DC, USA, 2012), SCC '12, IEEE Computer Society, pp. 821–826. URL: http://dx.doi.org/10.1109/SC.Companion.2012.115, doi:10.1109/SC.Companion.2012.115. 2

[MSU*16] MORELAND K., SEWELL C., USHER W., T. LO L., MEREDITH J., PUGMIRE D., KRESS J., SCHROOTS H., MA K. L., CHILDS H., LARSEN M., CHEN C. M., MAYNARD R., GEVECI B.: Vtk-m: Accelerating the visualization toolkit for massively threaded architectures. *IEEE Computer Graphics and Applications 36*, 3 (May 2016), 48–58. doi:10.1109/MCG.2016.48. 2

[NVI07] NVIDIA CORPORATION: *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. NVIDIA Corporation, 2007. 1, 2

[Rei] REINDERS J.: *Intel Threading Building Blocks*. 1, 2

[SGS10] SHI G., GOHARA D., STONE J. E.: Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in Science & Engineering 12* (2010), 66–73. doi:doi.ieeecomputersociety.org/10.1109/MCSE.2010.69. 2

[SH95] STALLING D., HEGE H.-C.: Fast and resolution independent line integral convolution. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1995), SIGGRAPH '95, ACM, pp. 249–256. URL: http://doi.acm.org/10.1145/218380.218448, doi:10.1145/218380.218448. 2

[SK97] SHEN H.-W., KAO D. L.: Uflic: a line integral convolution algorithm for visualizing unsteady flows. In *Visualization '97., Proceedings* (Oct 1997), pp. 317–322. doi:10.1109/VISUAL.1997.663898. 1, 2

[SK98] SHEN H.-W., KAO D. L.: A new line integral convolution algorithm for visualizing time-varying flow fields. *IEEE Transactions on Visualization and Computer Graphics 4*, 2 (Apr 1998), 98–108. doi:10.1109/2945.694952. 1, 2

[vW02] VAN WIJK J. J.: Image based flow visualization. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 745–754. URL: http://doi.acm.org/10.1145/566570.566646, doi:10.1145/566570.566646. 2