Improved Post Hoc Flow Analysis Via Lagrangian Representations

Alexy Agranovsky[*,†], David Camp[†], Christoph Garth[§], E. Wes Bethel[†], Kenneth I. Joy[*], and Hank Childs[†,‡]

University of California, Davis[*]
Lawrence Berkeley National Laboratories[†]
University of Oregon[‡]
University of Kaiserslautern[§]

ABSTRACT

Fluid mechanics considers two frames of reference for an observer watching a flow field: Eulerian and Lagrangian. The former is the frame of reference traditionally used for flow analysis, and involves extracting particle trajectories based on a vector field. With this work, we explore the opportunities that arise when considering these trajectories from the Lagrangian frame of reference. Specifically, we consider a form where flows are extracted *in situ* and then used for subsequent *post hoc* analysis. We believe this alternate, Lagrangian-based form will be increasingly useful, because the Eulerian frame of reference is sensitive to temporal frequency, and architectural trends are causing temporal frequency to drop rapidly on modern supercomputers. We support our viewpoint by running a series of experiments, which demonstrate the Lagrangian form can be more accurate, require less I/O, and be faster when compared to traditional advection.

Keywords: flow visualization, high-performance computing, compression, particle advection, pathline interpolation

1 INTRODUCTION

To achieve ever-higher levels of computational power, architects designing cutting edge supercomputers are forced to make difficult tradeoffs between networking capabilities, I/O capabilities, and memory per node, among other concerns. These tradeoffs are the result of finite budgets; architects play a zero-sum game where allocating extra dollars for one area requires reducing the dollars for another. Over the last decade, architects have increasingly opted toward reduced I/O capabilities. While I/O bandwidth is still increasing on almost every new supercomputer, it is often not keeping pace with their abilities to generate data.

For the most part, decreased I/O capabilities (relative to compute power) are *not* catastrophic for the simulation ecosystem. Simulation codes can drop the frequency with which they store their state — time slices — to disk, effectively enacting a defense mechanism that ensures the total time they spend doing I/O remains acceptable. The result, however, is that the temporal frequency of the data is reduced. The effect on subsequent analysis of this simulation data, often done with visualization, varies. It is sometimes largely unaffected, although it can greatly affect the quality of the results.

One strategy to counter-act temporal sparsity is to use *in situ* processing. Rather than the traditional *post hoc* model, where visualization and analysis codes read time slices that a simulation has stored to disk, *in situ* processing eschews the disk altogether, and does its processing as the simulation produces data. *In situ* processing has become increasingly popular, with many successful usages in recent years [4, 14, 16, 26, 27]. One of the advantages of *in situ* processing is that it can access all of the simulation data, which has never previously been possible with *post hoc* analysis. Phrased another way, where supercomputer trends are leading to temporal sparsity, *in situ* processing allows for dramatic increases in temporal frequency, equal to that accessible in the simulation code itself.

In this paper, we explore particle advection techniques from this context of temporal sparsity. Particle advection — moving a particle so that its displacement is tangent to a vector field — is a foundational technique for flow visualization, serving as the basis for pathlines, line integral convolution [2], finite-time Lyapunov exponents [8], streamsurfaces [11], and many other algorithms. Conceptually, particle advection assumes access to the complete spatio-temporal data; when faced with increasing temporal gaps, the resulting interpolations can have large errors, which in turn deviates the trajectory of a particle from its correct path.

We believe that the inaccuracies resulting from temporal sparsity require investigation into new techniques. With this work, we draw inspiration from fluid mechanics, which considers two frames of reference for an observer watching a flow field: Eulerian and Lagrangian. With the Eulerian frame of reference, the observer is at a fixed position and watches flow go by. This is the frame of reference traditionally used for advection in visualization. With the Lagrangian frame of reference, the observer is attached to a particle and moves through space and time. The concept of the Lagrangian frame of reference can be applied to visualization by taking a basis of known trajectories (Lagrangian flows), and then interpolating new particle trajectories from this basis.

We believe the Lagrangian paradigm is well-suited to the emerging supercomputing constraints that are driving the usage of *in situ* processing. Whereas reduced temporal frequency causes the accuracy of traditional particle advection techniques to suffer, a Lagrangian representation can be calculated *in situ*, meaning that their accuracy does not need to decrease. Further, where traditional particle advection techniques often have to pay significant I/O costs to process many time slices with unsteady flow, Lagrangian methods are stored temporally, meaning that unsteady flow analysis can be performed by reading much less data (i.e., just the necessary flows from the Lagrangian basis). Finally, Lagrangian analysis can throttle the amount of data stored, by reducing its basis of Lagrangian flows. While this reduction comes at a cost of accuracy, it can still be more accurate than the traditional approach when faced with temporal sparsity. To sum, when compared to traditional processing techniques, Lagrangian analysis has the potential to be more accurate, faster, and require less data stored on disk.

With this work, we describe a two-phase method that operates both *in situ* and *post hoc*. In the first phase, a Lagrangian basis of flows is extracted *in situ* and stored to disk. In the second phase, new particle trajectories can be explored *post hoc* by interpolating from the first phase's known trajectories. After describing the al-

[*]A. Agranovsky and K. I. Joy are with University of California at Davis. Email: aagranovsky@ucdavis.edu, joy@cs.ucdavis.edu

[†]D. Camp and E.W. Bethel are with Lawrence Berkeley National Laboratory. Email: dcamp@lbl.gov, ewbethel@lbl.gov

[‡]H. Childs is with the University of Oregon and Lawrence Berkeley National Laboratory. Email: hank@uoregon.edu

[§]Christoph Garth is with University of Kaiserslautern, Germany. Email: garth@cs.uni-kl.de

IEEE Symposium on Large Data Analysis and Visualization 2014
October 9–10, Paris, France
978-1-4799-5215-1/14/$31.00 ©2014 IEEE

gorithm in more depth (Section 3), we summarize our hypotheses about opportunities compared to traditional advection (Section 4), and then describe an experiment to evaluate its efficacy (Section 5). Our findings (Section 6) demonstrate that the potential outcomes motivating our study are realized: it is more accurate, requires less storage, and improves performance.

2 Related Work

2.1 Traditional Advection

Particle advection is a technique for calculating the trajectory a particle follows in a flow field. McLouglin et al. recently surveyed the state of the art in flow visualization [15], and the large majority of techniques they described incorporate particle advection. Advection assumes access to a vector field, i.e., a continuous function over a four-dimensional domain. If x is a spatial location of a point and t is a time, then the vector field v maps the tuple (x,t) to its velocity, as $v(x,t)$. For some approaches, visualization techniques focus on the special case of stationary flows whose vector fields does not vary over time ("steady state"). With this paper, our focus is on the general case: transient flows where the vector fields are time-varying ("unsteady state").

Advection constructs integral curves, which are continuous functions tangential to the vector field. The curves are solutions to an ordinary differential equation, and, for an integral curve I, can be represented as:

$$\frac{d}{dt}I(t) = v(I(t),t) \tag{1}$$

where $I(t_0) = x_0$, for a seed point at time t_0 and location x_0. An integral curve – commonly referred to as a pathline – encodes the trajectory of a single mass-less particle, which in turn gives insight into the flow behavior in the area surrounding the particle's path. Further, when considering many integral curves throughout the spatial domain, the entire flow field starts to be revealed, and is ultimately defined in terms of a *flow map*. A flow map denotes the movement of mass-less particles with respect to a vector field over a given time interval, as depicted in Figure 1. When a flow is described entirely by its flow map (and not by a vector field as with traditional advection), then this representation is referred to as Lagrangian.

2.2 Storing Flow Field Data

One strategy for addressing I/O constraints — while within the Eulerian frame of reference — is to compress time-varying vector fields as they are generated. Lodha [13] provided a "knob" that compressed similar vectors into one vector, covering a larger area. Theisel et al. [24] collapsed critical points and reduced the problem to mesh reduction. Later, the same authors provided a threshold to distinguish important features for filtering [23]. These works tackled compression by targeting individual time steps. Tong et al. [25] took a different approach by collapsing N number of simulation steps into a smaller, K number of time steps, highlighting temporal features of the data set. With our work, we examine the advantages that Lagrangian methods offer within the simulation stage for CFD, by way of decreasing the disk space required to store the flow field while maintaining high accuracy.

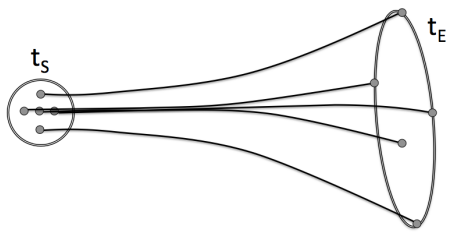

Figure 1: Visual flow map example. The figure shows the mapping of five particles from start position (within the circle labeled t_S) to end position (labeled t_E).

2.3 Lagrangian Methods for Flow Visualization

Lagrangian methods have had a significant presence within the flow visualization community in the last decade. The most notable has been the introduction of Lagrangian Coherent Structures (LCS) by Haller et al. [7,9], which illuminate distinct salient features within a flow field based on the calculation of stable and unstable manifolds. Embraced by the fluid dynamics community, focus turned to the acceleration of these computations for two-dimensional and three-dimensional flows through GPU acceleration [6], adaptive mesh refinement over grid data structures [5, 19], grid advection [20], and the interpolation over sparse particles [1].

A more recent development in Lagrangian methods has been to incorporate the Lagrangian view into the standard Eulerian representation of a flow field. Jobard et al. [12] presented a Lagrangian-Eulerian advection scheme which incorporated forward advection with a backward tracing Lagrangian step to more accurately shift textures during animation. Salzbrunn et al. delivered a technique for analyzing circulation and detecting vortex cores given predicates from pre-computed sets of streamlines [22] and pathlines [21]. In both cases, a dense sampling of the flow field is guaranteed by initially seeding at the center of each voxel on a regular grid and adding particles over time to fill gaps within the field (defined by a user set distance metric). Any particles seeded after the initial time step are also advected backwards to t=0. The authors note that this pre-processing step may take hours/days depending on the data set.

Hlawatsch et al. [10] use a hierarchical scheme to decrease the number of integration steps by constructing longer integral lines from previously computed partial solutions. They discuss the notion of pre-computing a set of Lagrangian-based trajectories and optimally choosing which of the trajectories to incorporate based on advection length. This work is most similar to our own, since it considers the projection of particles, rather than building integral curves through a series of advection steps. Their work focuses on utilizing temporal hierarchies of pathlines that overlap in time as well as effective usage of a GPU, but does not center as heavily on selection of flows, interpolation, improved accuracy, or the context of *in situ* calculation. Our work focuses much more heavily on the latter four and also provides the context of the Lagrangian frame of reference for conveying the underlying technique. Further, while their main focus is on increasing performance, our work aims at improved performance, greater accuracy, and reduced I/O when compared with traditional advection.

3 Lagrangian-Based Method

Our approach has two distinct phases.

In the first phase, we extract a basis of known pathlines. This phase occurs *in situ*, meaning that the extraction module can access all of the spatio-temporal data. This property is particularly helpful in a Lagrangian setting, since the Lagrangian representation captures particle behavior over an interval of time. This contrasts with the traditional technique, which relies on storing time slices, and thus can less readily make effective use of the extra temporal resolution from *in situ* processing. The *in situ* extraction of basis pathlines and storage to disk are discussed in Section 3.1.

In the second phase, we calculate arbitrary integral curves, that is, the trajectories from arbitrarily seeded particles. This phase is run *post hoc*, i.e., it enables exploration of the flow field. The only trajectories calculated are those that a user would request. The integral curves are calculated by interpolating from the first phase's basis pathlines, and this process is described in Section 3.2.

The Lagrangian method does not re-construct the original flow map, M. Instead, it constructs a new flow map, M', although the only flows from M' that are actually calculated are those requested by users. If M' is substantially similar to M, then the Lagrangian method has done a good job reconstructing the field; if it is very different, then it has done a poor job. We refer to the pathlines from the first (*in situ*) phase as basis pathlines, since any flow in M' can be reconstructed from them in the second (*post hoc*) phase — they span the space of reconstructed flows.

3.1 Extracting a Basis of Flows From the Simulation

Simulation codes advance in time in discrete steps. Restated, the simulation's notion of time will advance from time T to time $T+\Delta$, typically immediately after a linear solve. We refer to one advancement in time as a *cycle*. At the end of some cycles, simulations save out data to disk. We distinguish these cycles with their own term: *file cycle*.

The data stored to disk is frequently called a "dump," "restart dump," "time slice," or "checkpoint." We do not use these terms, since they have the connotation of saving the instantaneous state of the simulation. That description is appropriate for traditional advection, but the Lagrangian-based method requires data that corresponds to an interval of time (as opposed to a slice of time). Figure 2 illustrates a notional simulation code advancing in time and the files it saves for both methods.

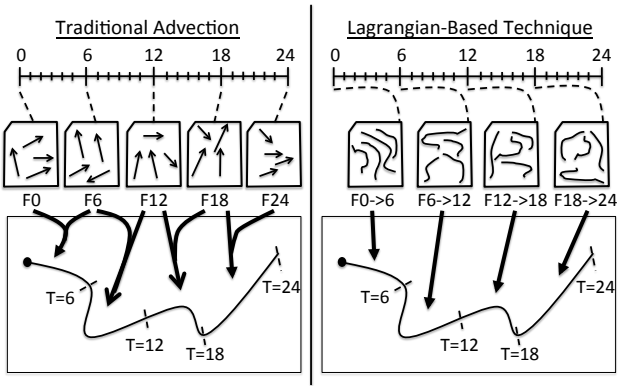

Figure 2: Comparison of traditional advection and Lagrangian advection. In both cases, the example simulation code runs for 24 cycles and outputs data every 6 cycles. With the traditional method, the output is a "snapshot" in time of the simulation's current state (denoted "F0" for the file containing the velocity at time 0, and so on) and advection occurs by solving ordinary differential equations and doing spatial and temporal interpolation. With the Lagrangian method, the output is a set of Lagrangian flows — the pathlines that form the basis flows — and *post hoc* analysis occurs by interpolating positions between the Lagrangian flows. The bottom portion shows a notional path that a particle may follow and shows what data is needed to perform advection. While the trajectories are identical between the two methods in this example, they will likely be different in practice. The traditional method has increasing error when the temporal frequency drops, while the Lagrangian method has increasing error when the number flows in its basis shrinks.

Another important difference for the Lagrangian-based method is that its *in situ* code must run every cycle, so it can advance the particles for the current time step. This contrasts with traditional advection, where the simulation does not need to incur any extra overhead for visualization purposes on non-file cycles.

Table 1 summarizes the differences between the approaches.

There are a variety of ways to extract a basis of pathlines from the simulation. Selecting a "good" basis will lead to higher accuracy, while selecting a "bad" basis will lead to lower accuracy. In Section 3.1.1 and Section 3.1.2, we describe the method we used in this study, optimized for minimizing storage costs. Future work involves considering other bases of pathlines and how they affect the overall accuracy of the reconstructed flow map.

	Traditional advection	Lagrangian-based method
Files represent	Time slice	Time interval
Files contain	Vector fields	Particle trajectories
Reducing I/O	Less time slices	Less particles
Increasing accuracy	More time slices	More particles
Work required during simulation	None	Advecting particles
Memory required for method	None	Storing trajectories

Table 1: Summarizing the differences from the simulation code's perspective between traditional advection and the Lagrangian-based method. Although the Lagrangian-based method has increased overhead from running *in situ*, we hypothesize that it has many advantages as well, and discuss those advantages in Section 4.

3.1.1 Seeding the Pathline Basis

The controls for choosing our pathline basis were (i) where and when we placed particle seeds and (ii) how long we allowed a particle to exist. For our study, we chose to seed particles along a uniform grid. The particle duration was set as the time elapsed between file cycles, meaning the duration was equal for every particle (rather than letting it vary per particle). At the end of a file cycle, position information is written to disk, the calculated curves are destroyed, and a new set of particles is seeded (again along a uniform grid), with advection resuming at the next cycle. The process is repeated for the length of the simulation.

While we chose short durations for our study, longer durations also have merit. As observed by Hlawatsch et al. [10], a particle's trajectory has greater accuracy when following a longer trajectory versus a series of shorter trajectories, arguably due to the approximation error that accumulates in the latter. However, longer trajectories create opportunities for new pitfalls. If new particles are not frequently introduced, then longer-lived integral curves may move away from certain regions over time, creating regions of the data set with little to no coverage. This would likely result in poor reconstructed flow maps in these regions when doing *post hoc* analysis. Further, our mechanism for choosing particles was simple: there was no need to identify regions that have too few integral curves (and thus more need to be added) or too many (and thus computation was being wasted). With short-lived particles, these concerns were mostly mitigated. We also note that the simplicity led to reduced storage; if new integral curves are created while previously-born curves remain, an identification number must be assigned with each curve to track its information across multiple file cycles. Thus, storing long-lived integral curves adds an integer per curve, having an adverse affect on disk space and complicating implementation. Along the lines of saving on storage, we aligned the integral curves along a uniform grid because this approach eliminates the need for explicitly storing the starting position information.

3.1.2 Storing the Pathline Basis

The minimal way to represent an integral curve is with two positions; a start position and an end position. (This form is likely acceptable for short-lived integral curves, but not for long-lived integral curves.) Because we have chosen to align the integral curves with a regular grid, only the end position must be stored, at a cost of 3 floats or 12 bytes per curve. With this type of file formatting, the

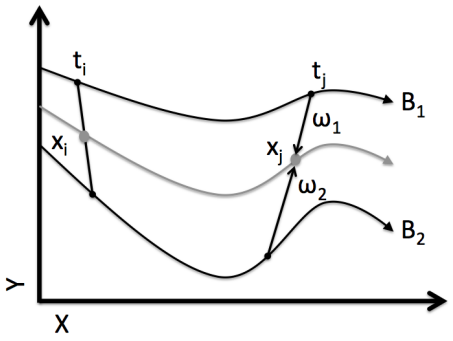

Figure 3: Interpolation using the Lagrangian basis. Position values of pathline x are interpolated from basis pathlines B_1 and B_2, using weights ω_1 and ω_2, respectively. The weights are calculated based on the location of all pathlines at t_i and then applied at t_j to approximate x_j.

Lagrangian representation now uses the exact same amount of storage per file cycle as required by the traditional method when tracing one particle for every grid point in the mesh. (When tracing one particle for every eight grid points, it would take one eighth of the storage, etc.) Again, a file for the Lagrangian method represents an interval of time. This contrasts with the traditional method, which, to evaluate velocities at times other than the time slices stored at file cycles, needs to read the two vector fields that bracket the desired time and interpolate between them. Over an entire time series, the formatting difference translates to one less file required for the Lagrangian representation.

3.2 Post Hoc Exploration — Interpolating New Integral Curves From the Lagrangian Basis

During *post hoc* analysis, the Lagrangian representation relies on the interpolation of position information to create pathlines, where this interpolation between position values can be achieved through a variety of techniques.

For our work, we have chosen to use barycentric coordinates to interpolate between integral curve start and end positions, because a tetrahedron is the minimum convex hull that can be formed around any point in three-dimensional space. Barycentric coordinates define a position with respect to the positions of the vertices of the tetrahedron they describe:

$$P = AW_A + BW_B + CW_C + DW_D \tag{2}$$

where A, B, C, and D are vertices of the tetrahedron and W_i represents the volume coordinates, subject to the constraint $W_A + W_B + W_C + W_D = 1$.

To begin the interpolation process, any arbitrarily placed seed particle must first be enclosed by a tetrahedron. Instead of converting the global grid into a tetrahedral mesh, a standard template is used to describe how a rectangular cell breaks down into tetrahedron, with the enclosing tetrahedron being identified on the fly.

Multiple methods exist for breaking a rectangle into tetrahedron [17] and for our work we have chosen to cut each cell into 5 tetrahedron. Once the appropriate tetrahedron is located, the particle's location is written in terms of the tetrahedron vertices, representative of integral curve start positions. The volume coordinate ω_i determines the weighting that will be applied to the respective integral curve end positions to interpolate the next position of the particle. Figure 3 provides a visual two-dimensional example where two basis pathlines are interpolated to find the position of pathline x at time t_j.

4 Comparing with Traditional Advection

We hypothesize that the Lagrangian-based method has the potential to improve on traditional advection with respect to accuracy, I/O costs, and performance.

4.1 Accuracy

While the accuracy of traditional advection erodes as temporal frequency drops, the accuracy of the Lagrangian-based method is unaffected. This is because the Lagrangian-based method is calculated *in situ* and thus, utilizes information from all time slices of the data set.

When doing the *post hoc* interpolation of a new flow, flow field information will not be explicitly available for all cycles. That said, the Lagrangian nature of the method reflects the flow behavior in the integral curve positions. The Lagrangian representation maps the paths that particles take between file cycles, meaning that, although the path may be wrong for times between the file cycles, it will be highly accurate for times corresponding to file cycles; furthermore, these (accurate) positions will be used to make the next steps forward. This contrasts with traditional advection, where errors compound more easily. Figure 4 conveys the two different approaches creating integral curves from two particles seeded at the same location, marked L for Lagrangian and A for advection. They are compared against a pathline B, which has been advected using the vector field of each cycle in the simulation.

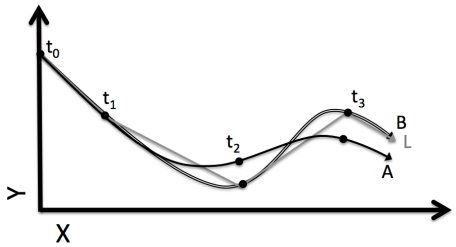

Figure 4: Notional pathline comparison between the Lagrangian representation and the traditional approach for particles seeded at the same location. Pathline A represents an integral curve advected through vector fields available only at file cycles, while pathline B uses all simulation vector fields for advection over the same time frame. Pathline L, interpolated from the Lagrangian basis, is able to approximate pathline B with higher accuracy than pathline A.

4.2 I/O Costs

If TS is the number of time slices and VF is the bytes associated with storing the vector field for one time slice, then the memory required for traditional advection is $TS \times VF$. The size of VF is typically fixed, unless subsampling or other compression techniques are used. Similarly, TS is typically set by the simulation code. So, the total memory used for traditional advection is typically fixed and outside the control of the visualization. Of course, time slices are sometimes generated just for visualization purposes and, in this case, TS can be treated as variable.

If $TRAJ$ is the number of bytes to store a particle trajectory of a single particle for one time interval, NP is the number of particles, and INT is the number of time intervals, then the memory required for the Lagrangian-based method is $TRAJ \times NP \times INT$. INT, like TS for traditional advection, is typically set by the simulation code. $TRAJ$ is also fixed. However, NP is a new control. If I/O costs are prohibitive — and evidence is accumulating that this is increasingly the case — then the number of particles can be reduced. The positive effect of this reduction is an I/O savings. The negative effect, however, is that the basis of Lagrangian flows is reduced and therefore the accuracy of the *post hoc* reconstructions will decrease.

4.3 Performance

Here we refer to the performance of *post hoc* calculation of particle trajectories for flow-based visualization techniques. With traditional advection, the vector fields are usually arranged such that all data from one time slice is grouped together. This is not optimal for unsteady-state analysis, since data from many locations need to be read. Worse, a typical implementation reads the entire data field for each time slice, i.e., reads in a time slice, evaluates where a particle is located within that time slice, advects, discards the time slice, and repeats for future time slices. With the Lagrangian-based method, the data is arranged by time. As a result, only the relevant flows from the basis need to be read in to calculate a particle trajectory.

5 Study Overview

We devised a study to test our hypotheses about the benefits of the Lagrangian-based method.

5.1 Configurations

The study was designed to provide coverage over three factors:

1. Data reduction (7 options)
2. File cycle intervals (4 options)
3. Data sets (3 options)

We ran the cross-product, meaning $7 \times 4 \times 3 = 84$ tests overall.

5.1.1 Data Reduction

Our data reduction is based off the vector field grid size for one cycle, i.e. $\frac{1}{1}$ is the same size as the vector field and $\frac{1}{64}$ is one sixty fourth the size, with a number of particles seeded for every grid point in the equivalent mesh . We considered seven reduction factors: $\frac{1}{1}$, $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$, $\frac{1}{32}$, and $\frac{1}{64}$.

5.1.2 File Cycles

The number of cycles between file saves can vary between simulations and even with the same simulation on different computer systems. We considered four different intervals for file cycles: 20, 40, 80, and 160.

5.1.3 Data Sets

We used the following three data sets to evaluate our method:

Arnold-Beltrami-Childress (ABC) - This three-dimensional time-dependent variant of the ABC analytic vector field is from the field of dynamical systems theory. The ABC vector field follows the parameterization of A=$\sqrt{3}$, $B = \sqrt{2}$, and $C = 1$. This dataset is simulated for 3000 cycles, with each cycle consisting of a regular grid with dimensions $256 \times 256 \times 256$.

Double Gyre - This two-dimensional time-dependent double gyre velocity field has become a common analytical test problem for techniques involving Lagrangian flow fields. The flow field consists of two counter-rotating gyres with a time dependent perturbation. This dataset is simulated for 3000 cycles, with each cycle consisting of a regular grid with dimensions 512×256.

Jet - The three-dimensional Jet dataset was created using the Gerris Flow Solver [18]. It is a simulation of a high-speed jet entering a medium at rest. This dataset contains 2000 cycles, with each cycle consisting of a regular grid with dimensions $260 \times 520 \times 260$.

5.2 Output

Seven different Lagrangian representations — one for each reduction factor — are created for each file cycle interval. Also, we created an Eulerian representation (vector field) for every cycle. This was used to obtain the "ground truth" pathlines, i.e., it allowed us to calculate trajectories with all spatio-temporal data.

We denote the outputs as:

- V_C - Eulerian representation (vector field) from every cycle.
- V_{FC} - Eulerian representation (vector field) from each file cycle.
- $L_{\frac{1}{X}}$ - Lagrangian representation for each data reduction value from each file cycle, with X being the reduction factor.

5.3 Error Evaluation

A set of pathlines is created using the Eulerian and Lagrangian representations; they are denoted by P_C, P_{FC}, and $P_{\frac{1}{X}}$. The pathline sets P_C and P_{FC} use a fourth-order Runge-Kutta scheme [3] for advection and the $P_{\frac{1}{X}}$ sets use barycentric coordinates for interpolation. For the purpose of quantifying accuracy, P_C is assumed to be perfectly accurate. The pathlines from all other sets are measured against the pathlines of P_C, calculating the average Euclidean distance between all pathlines within the set. The average distance is then normalized by the cell size of its source data set, to rule out differences between pathlines stemming from the simulation's spatial volume.

The number of pathlines in P_C, P_{FC}, and $P_{\frac{1}{X}}$ were large, so that we could evaluate both the accuracy of the reconstruction, and the time it took to perform the reconstruction. For the ABC data set, the number of pathlines was 1.1 million, for the Jet data set, 2.9M, and for the Double Gyre, ten thousand.

5.4 Runtime Environment

We performed all tests on Hopper, a Cray XE6 at Lawrence Berkeley's NERSC supercomputing center. Each node contains two twelve-core AMD MagnyCours processors running at 2.1 GHz. All of our simulation tests used 3,072 cores.

6 Results

In our results, we focus on a comparison between pathlines created from a Lagrangian representation and those advected through the traditional approach, with respect to accuracy and storage costs (6.1), and also performance (6.2).

6.1 Accuracy and Storage Costs

The number of basis flows is a key control ("knob") for simulation scientists during the *in situ* extraction phase. Scientists wanting higher accuracy can request more basis pathlines (thus reducing potential storage savings), or they can request fewer basis pathlines (thus reducing potential accuracy). Ergo, storage and accuracy are in tension, and so we present the results for the two together, acknowledging how one rises while the other falls.

As mentioned in Section 5.3, we evaluate error using the Euclidean distance between temporally equivalent positions on the pathlines for sets $P_{FC}, P_{\frac{1}{1}} - P_{\frac{1}{64}}$ against those in P_C. In essence, the accuracy evaluation measures how closely the *post hoc* portion of the method is able to approximate the original flow map. Further, our accuracy measure reflects evaluations over a large number of pathlines distributed throughout the problem domain, making us confident that it captures underlying trends.

Figure 5 depicts accuracy measurements for all of the data sets. The x-axis denotes simulation time (in units of cycles), as the accuracy varies throughout the simulation and so the values for different cycles in the simulation need to be plotted separately. The y-axis is based on our accuracy metric, i.e., comparing pathlines created with the Lagrangian approach to those created using the traditional approach. However, the y-axis does not plot the absolute value of the metric. Instead, each plot is normalized by the accuracy of the traditional approach (i.e., P_{FC}), which allows for quick comparisons with the Lagrangian representation. Specifically, y-axis values above (or below) one indicate how many times more (or less) accurate the Lagrangian approach is when compared to the traditional approach. For visual clarity, the horizontal line $y = 1$ is

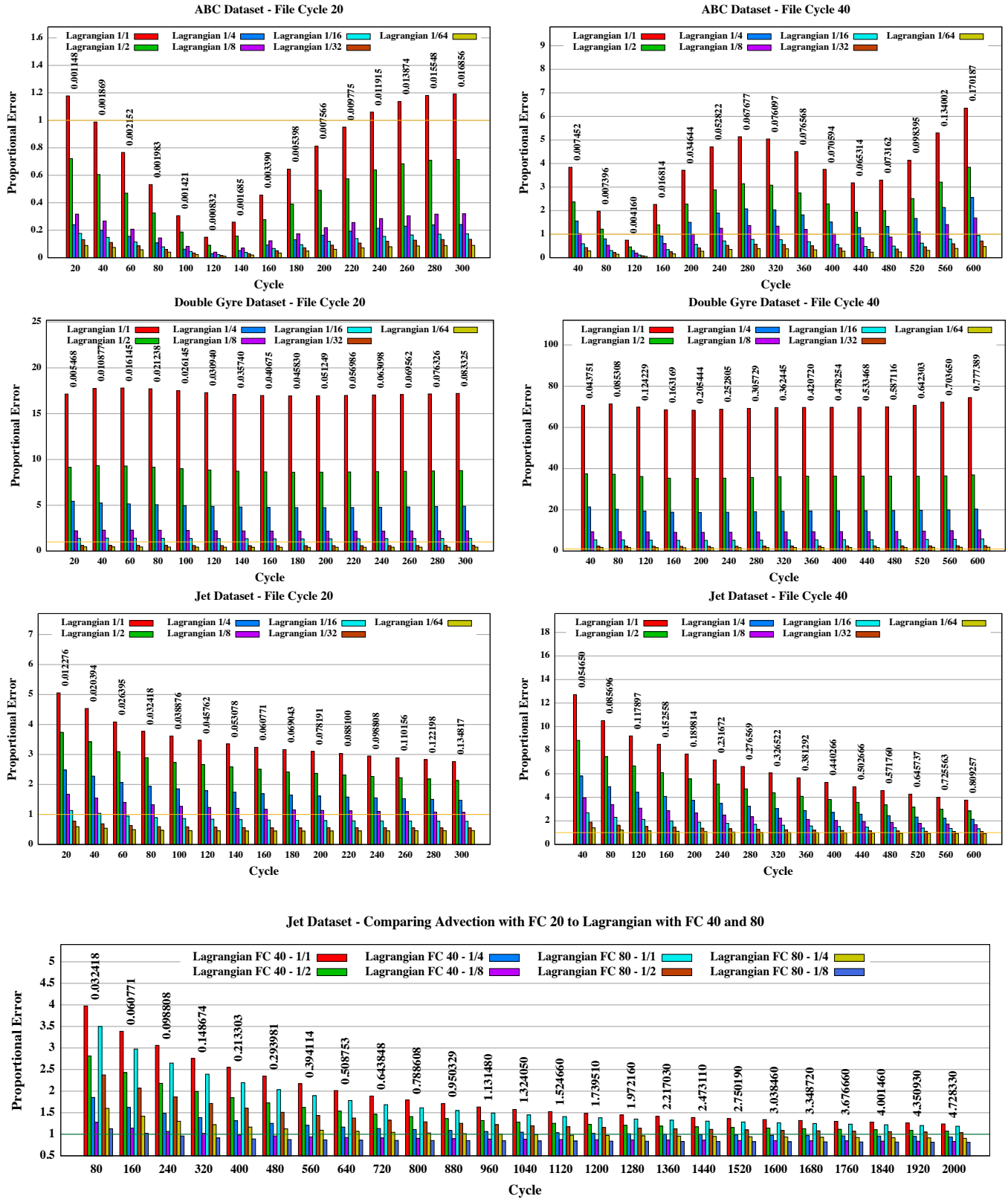

Figure 5: Each of the seven sub-figures plot Lagrangian error for a data set and file cycle interval. The error varies as the simulation evolves, so the error is plotted with respect to the simulation cycle time (on the x-axis). A grouping of bars for a given cycle represent the error for different reduction factors for the Lagrangian approach during that simulation time. These errors are normalized by the average error for the traditional approach, and this average error is displayed numerically above the grouping. The horizontal line $y = 1$ denotes where the Lagrangian and traditional approaches produce the same error. Bars above that line are more accurate and bars below that line are less accurate.

highlighted in each graph. Each cycle has a group of seven bars, denoting the normalized error for the sets $P_{\frac{1}{1}} - P_{\frac{1}{64}}$, from left to right, respectively. Further, the average error for the traditional approach is displayed numerically above that cycle's seven bars.

6.1.1 Arnold-Beltrami-Childress

The top row of Figure 5 shows the ABC data set at file cycles twenty and forty. A particularly noteworthy feature of both graphs is the sinusoidal behavior of the error, which is largely due to the sinusoidal ABC analytical function. Due to the absence of velocity information for every simulation cycle, the traditional advection method produces pathlines of a markedly different sinusoidal frequency than the ground truth. In contrast, the Lagrangian representation allows for pathlines to retain and closely match the frequency of the ground truth; this disparity causes the sinusoidal behavior of the error bars shown in the error analysis. Thus, even if the Euclidean distances between the traditionally advected pathlines and those of the ground truth are closer to one another than the Lagrangian methods, they are nonetheless following a completely different velocity field. While it is well understood that the absence of all spatio-temporal data affects accuracy with the traditional technique, we feel this example particularly underscores how inaccurate this approach can become.

There are many instances in this data where the Lagrangian approach uses less storage and is still more accurate. For example, with file cycle forty and one eighth of the storage, the Lagrangian technique is still more accurate than the traditional technique more than half the time.

6.1.2 Double Gyre

The second row of Figure 5 displays the error from the Double Gyre data set. The majority of pathlines rotate about either of the two gyres and the average error increases constantly over time. This data set illustrates how drastically the flow field can change within a small space. Resultantly, when fewer basis flows are used, there is a sharp decline in accuracy and the range of errors is spread out. For example, at a file cycle of twenty, pathline set $L_{\frac{1}{1}}$ is over fifteen times more accurate than V_{FC}, $L_{\frac{1}{2}}$ is almost ten times as accurate, and $L_{\frac{1}{4}}$ is more than five times as accurate.

The Double Gyre exemplifies that in turbulent areas, accurate flow field information can be obtained even with high reductions. In the case of $L_{\frac{1}{16}}$, the Lagrangian approach creates more accurate pathlines than the traditional approach, even though the traditional approach uses sixteen times more storage. The Double Gyre chart for file cycle forty demonstrates that, as the file cycle increases, even fewer basis pathlines are needed to achieve higher accuracy than the traditional approach.

6.1.3 Jet Data

The bottom three charts of Figure 5 represent the error analysis for the Jet data set, in which a central jet shoots upwards from the bottom of the domain. The flow moves very quickly within the jet itself, and pathlines that are seeded within this space either leave the data set or begin to circulate between the jet and the boundary of the domain. As a result, as the simulation advances in time, the error levels off, as seen in the bottom three graphs of Figure 5. Regardless of file cycle size, pathlines interpolated using the Lagrangian approach are better able to capture the turbulent and fast-paced motion that occurs within the jet itself. This effect can be seen towards the beginning cycles of every chart.

For each data set presented, the Lagrangian representation is able to retain substantially better accuracy when compared to the traditional method, and at times, even with significantly less flow field information. As the file cycle increases, the difference becomes even more pronounced: the Lagrangian representation retains the original flow field at a much lower cost. With a file cycle of twenty, storing basis pathlines taking one eighth of the disk space as the traditional approach produces comparable accuracy. With a file cycle of forty, comparable accuracy can be achieved with only $\frac{1}{64}$th the information.

Jet Data Set		File Cycle 20		File Cycle 40	
Method	File Size (MB)	# of Files	Total Size (GB)	# of Files	Total Size (GB)
V_{FC}	402.3	101	**39.7**	51	**20.0**
$L_{\frac{1}{1}}$	402.3	100	39.2	50	19.6
$L_{\frac{1}{2}}$	201.1	100	19.6	50	9.8
$L_{\frac{1}{4}}$	100.6	100	9.8	50	4.9
$L_{\frac{1}{8}}$	50.3	100	**4.9**	50	2.5
$L_{\frac{1}{16}}$	25.1	100	2.5	50	1.2
$L_{\frac{1}{32}}$	12.6	100	1.2	50	0.6
$L_{\frac{1}{64}}$	6.3	100	0.6	50	**0.3**

Table 2: Disk space required to store flow field information from the simulation. The top row shows the traditional method. Subsequent rows show Lagrangian representations with various numbers of basis flows (and thus sizes). The bold values correspond to the amount of disk space necessary for a Lagrangian representation to create pathlines of equivalent accuracy as the traditional method. As the file cycle increases, the Lagrangian representation needs less storage space to match the accuracy of the traditional approach.

The final (bottom) plot in Figure 5 compares the errors for the Lagrangian representations for file cycles of forty and eighty. To facilitate comparison, the errors are normalized by the same denominator, specifically the traditional method with a file cycle of twenty. This contrasts with all other plots in this figure (which normalize the Lagrangian method at a given file cycle by the error for the traditional method at the *same* file cycle). The plots shows that the the Lagrangian approach creates more accurate pathlines, despite flow field information being stored less frequently. This is possible because *in situ* processing enables access to all spatio-temporal data, so the pathline basis can be advected using every simulation cycle, thereby retaining the temporal flow field regardless of file cycle. As a result, increasing the file cycle is much less detrimental for the Lagrangian approach when compared to the traditional approach.

Our tests successfully ran up to file cycles of one hundred and sixty. The results from this longer file cycle (as well as those from file cycles of eighty) exhibit the same patterns as the shorter file cycles. At a file cycle of twenty, the $L_{\frac{1}{16}}$ basis matches the accuracy of the traditional method; at a file cycle of forty, $L_{\frac{1}{64}}$ achieves comparable accuracy. The results we obtained for file cycles eighty and one hundred and sixty agreed with this trend. That is, higher file cycles appear to need fewer basis pathlines for equivalent accuracy.

Table 2 shows the storage space required to store the Jet data for the different methods. Once again, the finding is that the Lagrangian representation requires much less disk space to create pathlines of similar accuracy during the *post hoc* analysis phase. Our results only report the storage costs for Jet, since it is the largest of the three, and the trends are similar across the three data sets.

6.2 Performance

In this section, we analyze two aspects of performance. First, in Section 6.2.1, we consider the overhead for the *in situ* extraction of basis flows. Second, in Section 6.2.2, we consider the time needed to (*post hoc*) interpolate new pathlines from the Lagrangian basis and compare it to the traditional method.

Jet Data	FC 20 (s)	FC 40 (s)	FC 80 (s)	FC 160 (s)
$L_{\frac{1}{1}}$	0.554	0.535	0.587	0.656
$L_{\frac{1}{2}}$	0.272	0.283	0.306	0.351
$L_{\frac{1}{4}}$	0.163	0.145	0.155	0.179
$L_{\frac{1}{8}}$	0.092	0.082	0.083	0.096
$L_{\frac{1}{16}}$	0.053	0.047	0.048	0.052
$L_{\frac{1}{32}}$	0.033	0.029	0.030	0.031
$L_{\frac{1}{64}}$	0.024	0.021	0.022	0.022

Table 3: Time per cycle to extract the Lagrangian basis flows *in situ* for different file cycle intervals and reduction factors, measured in seconds.

	Jet		ABC		Double Gyre	
File	2.9M Pathlines		1.1M Pathlines		10K Pathlines	
Cycle	E (s)	L (s)	E (s)	L (s)	E (s)	L (s)
20	8.83	31.0	10.5	8.42	3.83	.056
40	4.89	15.3	5.83	4.49	1.98	.029
80	2.91	6.97	3.23	2.02	1.08	.014
160	1.67	4.18	1.77	.99	.617	.007

Table 4: Comparison of times, in seconds, for the traditional and Lagrangian methods to generate sets of pathlines for different data sets. The traditional method is denoted with an 'E' (for Eulerian) while the Lagrangian method is denoted with an 'L'. While the traditional method is faster in some cases, it had significantly more computational power to work with, as a result of our experimental configuration.

6.2.1 Simulation Overhead

Table 3 reports timings for the *in situ* extraction of basis flows from the Jet data set. (As Jet was the largest data set, the timings for other data sets were consistently smaller.) The table shows that, unsurprisingly, the time to extract the basis flows drops significantly as the number of flows extracted decreases. Also, when the interval between file cycles increases, efficiency drops, likely due to sinks in the flow field causing imbalance in the workload over the processors.

6.2.2 Creating Pathlines

We compare the computation time of creating pathlines during the *post hoc* phase. Again, the traditional approach uses Runge-Kutta 4 for numerical integration, while the Lagrangian approach uses interpolation over barycentric coordinates. The traditional approach is run on the same setup as the simulation, utilizing 3,072 cores. The Lagrangian approach, however, uses only one node with 24 threads, utilizing $\frac{1}{128}$th the power of the traditional run (meant to better exemplify a use case on a local machine). An equal number of pathlines are created for each analysis run, thereby all $L_{\frac{1}{x}}$ runs take a similar amount of time for interpolation with very slight variability due to neighbor location. We note that this configuration is not perfect: when interpolating few pathlines, fewer cores may be better (as efficiency may be higher), and, further, when interpolating many pathlines, the I/O costs can be amortized. That said, we believe the results presented demonstrate the performance advantages that the Lagrangian approach can yield.

Table 4 shows the pathline creation times for both methods over all data sets. For 2.9 million pathlines on the Jet data set, the longest time was 8.83 seconds, with computation time cut in half as the file cycle doubles. While a fraction of particles are seeded within the jet plume itself, the majority of particles for our test lie outside of the jet and remain close to their seed position as time advances. This creates a condition of fairly equal load balancing among the nodes during advection. The longest time to create equally seeded pathlines using the Lagrangian approach takes 31.0 seconds. Even though the interpolation has $\frac{1}{128}$th the computation power to draw upon, the computation time takes only between 2.5 to 3.5 times longer.

The ABC data set is susceptible to load imbalance. As a result, traditional advection of 1.1 million particles took longer than the advection of 2.9 million particles in the Jet data set. With the ABC data set, every area of the domain experiences high velocity and thus, particles are constantly being shipped across nodes. Consequently, the Lagrangian approach performs faster than the traditional method, with the longest time taking 8.42 seconds compared to 10.5 seconds for advection. As expected, the execution time for the Lagrangian approach is proportional solely to the number of pathlines to calculate: 1.1M pathlines on the ABC data set took consistently one third of the time as 2.9M pathlines on the Jet data set.

For the Double Gyre data set, with its small size, only 10,000 particles are calculated. The timings reflect the toll of communication costs, with the Lagrangian approach significantly outperforming the traditional method.

7 Conclusion and Future Work

We have introduced a method for *in situ* extraction of particle trajectories and subsequent *post hoc* exploration of flow, all from the Lagrangian frame of reference. This paradigm is a departure from traditional advection, which depends on saving time slices of data and assumes the Eulerian frame of reference. We hypothesized that the Lagrangian-based method would create opportunities for increased accuracy, reduced storage, and better performance. We then performed a series of experiments, designed to assess whether the Lagrangian-based method realized these opportunities. The results were compelling, and we believe they establish that this method has significant merit.

While our study indicates the potential utility of a Lagrangian-based approach, we understand that our method should be viewed as one instance of a larger family, and that other variations could lead to even better results. Specifically, we believe alternative approaches for selecting Lagrangian flows could improve accuracy and/or I/O savings. Further, we presented a scheme where Lagrangian flows were saved out at regular intervals. This is one way to accomplish our goal, but there are others. For example, flows could originate and terminate as needed, to ensure that each flow added to the basis is minimizing the total error. Finally, we believe comparisons with the work of Hlawatsch et al. [10], in terms of accuracy, storage, and performance, would be very useful. We plan to pursue these directions and others in the near future.

Acknowledgements

This work was supported by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231, and was funded in part by the Marie Curie Actions within the EU FP7 Programme under grant #304099. Hank Childs is grateful for support from the DOE Early Career Award, Contract No. DE-FG02-13ER26150, Program Manager Lucy Nowell. This research used resources of the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

REFERENCES

[1] A. Agranovsky, C. Garth, and K. I. Joy. Extracting flow structures using sparse particles. *Vision, Modeling, and Visualization 2011*, pages 153–160, 2011.

[2] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 263–270, New York, NY, USA, 1993. ACM.

[3] J. R. Cash and A. H. Karp. A variable order runge-kutta method for initial value problems with rapidly varying right-hand sides. *ACM Trans. Math. Softw.*, 16(3):201–222, Sept. 1990.

[4] N. Fabian, K. Moreland, D. Thompson, A. C. Bauer, P. Marion, B. r. Geveci, M. Rasquin, and K. E. Jansen. The paraview coprocessing library: A scalable, general purpose in situ visualization library. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 89–96. IEEE, 2011.

[5] C. Garth, F. Gerhardt, X. Tricoche, and H. Hagen. Efficient computation and visualization of coherent structures in fluid flow applications. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1464–1471, Nov 2007.

[6] C. Garth, G. Li, X. Tricoche, C. D. Hansen, and H. Hans. Visualization of coherent structures in transient 2d flows. *In Topology-Based Methods in Visualization*, Proceedings of the 2007 Workshop, 2007.

[7] G. Haller. Finding finite-time invariant manifolds in two-dimensional velocity fields. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 10(1):99–108, 2000.

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

[9] G. Haller and G. Yuan. Lagrangian coherent structures and mixing in two-dimensional turbulence. *Physica D: Nonlinear Phenomena*, 147(3-4):352 – 370, 2000.

[10] M. Hlawatsch, F. Sadlo, and D. Weiskopf. Hierarchical line integration. *Visualization and Computer Graphics, IEEE Transactions on*, 17(8):1148–1163, Aug 2011.

[11] J. P. M. Hultquist. Constructing stream surfaces in steady 3d vector fields. In *Visualization, 1992. Visualization '92, Proceedings., IEEE Conference on*, pages 171–178, Oct 1992.

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

[13] S. Lodha, J. Renteria, and K. Roskin. Topology preserving compression of 2d vector fields. In *Visualization 2000. Proceedings*, pages 343–350, 2000.

[14] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin. Flexible io and integration for scientific codes through the adaptable io system (adios). In *Proceedings of the 6th international workshop on Challenges of large applications in distributed enviro nments*, CLADE '08, pages 15–24, New York, NY, USA, 2008. ACM.

[15] T. McLoughlin, R. S. Laramee, R. Peikert, F. H. Post, and M. Chen. Over Two Decades of Integration-Based, Geometric Flow Visualization. In *EuroGraphics 2009 - State of the Art Reports*, pages 73–92, April 2009.

[16] K. Moreland, R. Oldfield, P. Marion, S. Jourdain, N. Podhorszki, V. Vishwanath, N. Fabian, C. Docan, M. Parashar, M. Hereld, et al. Examples of in transit visualization. In *Proceedings of the 2nd international workshop on Petascal data analytics: challenges and opportunities*, pages 1–6. ACM, 2011.

[17] G. M. Nielson. Tools for triangulations and tetrahedrizations. In *Scientific Visualization, Overviews, Methodologies, and Techniques*, pages 429–525, Washington, DC, USA, 1997. IEEE Computer Society.

[18] S. Popinet. Gerris: a tree-based adaptive solver for the incompressible euler equations in complex geometries. *J. Comput. Phys.*, 190(2):572–600, 2003.

[19] F. Sadlo and R. Peikert. Efficient visualization of lagrangian coherent structures by filtered amr ridge extraction. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1456–1463, Nov 2007.

[20] F. Sadlo, A. Rigazzi, and R. Peikert. Time-dependent visualization of lagrangian coherent structures by grid advection. *Topological Methods in Data Analysis and Visualization*, pages 151–165, 2011.

[21] T. Salzbrunn, C. Garth, G. Scheuermann, and J. Meyer. Pathline predicates and unsteady flow structures. *The Visual Computer*, 24(12):1039–1051, 2008.

[22] T. Salzbrunn and G. Scheuermann. Streamline predicates. *Visualization and Computer Graphics, IEEE Transactions on*, 12(6):1601–1612, Nov 2006.

[23] H. Theisel, C. Rossl, and H. P. Seidel. Combining topological simplification and topology preserving compression for 2d vector fields. In *Computer Graphics and Applications, 2003. Proceedings. 11th Pacific Conference on*, pages 419–423, 2003.

[24] H. Theisel, C. Rössl, and H.-P. Seidel. Compression of 2d vector fields under guaranteed topology preservation. *Computer Graphics Forum*, 22(3):333–342, 2003.

[25] X. Tong, T.-Y. Lee, and H.-W. Shen. Salient time steps selection from large scale time-varying data sets with dynamic time warping. In *Large Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on*, pages 49–56, Oct 2012.

[26] V. Vishwanath, M. Hereld, and M. Papka. Toward simulation-time data analysis and i/o acceleration on leadership-class systems. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 9–14, 2011.

[27] B. Whitlock, J. M. Favre, and J. S. Meredith. Parallel in situ coupling of simulation with a fully featured visualization system. In *Proceedings of the 11th Eurographics conference on Parallel Graphics and Visualization*, pages 101–109. Eurographics Association, 2011.