Noname manuscript No.
(will be inserted by the editor)

Compression-based Integral Curve Data Reuse Framework for Flow Visualization

Fan Hong · Chongke Bi · Hanqi Guo · Kenji Ono · Xiaoru Yuan

Received: date / Accepted: date

Abstract Currently, by default, integral curves are repeatedly re-computed in different flow visualization applications, such as FTLE field computation, source-destination queries, etc., leading to unnecessary resource cost. We present a compression-based data reuse framework for integral curves, in order to greatly reduce their retrieval cost, especially in a resource-limited environment. In our design, a hierarchical and hybrid compression scheme is proposed to balance three objectives, including high compression ratio, controllable error, and low decompression cost. Specifically, we use and combine digitized curve sparse representation, floating-point data compression, and octree space partitioning to adaptively achieve the objectives. Results have shown that our data reuse framework could acquire tens of times acceleration in the resource-limited environment compared to on-the-fly particle tracing, and keep controllable information loss. Moreover, our method could provide fast integral curve retrieval for more complex data, such as unstructured mesh data.

Xiaoru Yuan is the corresponding author.

Fan Hong and Xiaoru Yuan
Key Laboratory of Machine Perception (Ministry of Education), and School of EECS, Peking University
E-mail: {fan.hong, xiaoru.yuan}@pku.edu.cn

Chongke Bi
School of Software, Tianjin University
E-mail: bichongke@tju.edu.cn

Kenji Ono
Research Institute for Information Technology, Kyushu University, and Advanced Institute for Computational Science, RIKEN
E-mail: keno@cc.kyushu-u.ac.jp

Hanqi Guo
Mathematics and Computer Science Division, Argonne National Laboratory
E-mail: hguo@anl.gov

Keywords Flow Visualization · Integral Curves · Flow Lines · High Performance Visualization · Data Compression · Information Retrieval

1 Introduction

Particle tracing is the foundation of flow visualization. Typical examples include most of the geometry-based and texture-based flow visualization methods, source-destination queries, Finite-Time Lyapunov Exponent (FTLE) computation, etc. These visualization applications are all closely related with integral curves, which usually denote as pathlines and streamlines. Although computational- and data-intensive, it is necessary to compute integral curves in flow field to support various visualization tasks.

In this work, our focus is data reuse of integral curves. The main observation in this study is that streamlines and pathlines, as the intermediate data products in visualization pipelines, are usually computed on-the-fly and then discarded. This leads to a waste of computational resources in the re-analysis, especially for large and complex flow datasets. For example in FTLE analysis, when a user changes the parameters to previous settings, it requires re-computing the pathlines if they were discarded. In this case, the data need to be loaded and the seeds are traced over and over again.

The main obstacle for the reuse of integral curves is their volume, which could be several magnitudes larger than the input flow field data. For example, if we trace pathlines from every grid point, and each pathline contains 10^3 sample points, the overall data scale of the pathlines is 10^3 times larger than the input data. The overwhelmingly large data scale makes it prohibitive for the naive data reusing scheme.

For large scale data which requires scalable and parallel solutions, the overhead of re-computation is even larger due to the high cost of computation, I/O, and communication bandwidths. Intricate data partitioning and load balancing problems are also involved in for improving scalability. The limited allocation on supercomputers also makes it even more unaffordable for the waste. In this situation, to enable fast integral curve retrieval for large-scale flow visualization in a resource-limited environment is more important. In addition, the data grid complexity also makes it uneconomical to re-compute integral curves instead of reusing from previous runs. The point locating problems and sophisticated interpolation schemes result in even higher computation cost. In our cases, we demonstrate the successful story of integral curves reuse for datasets with hybrid-sigma curvilinear mesh and unstructured mesh.

In this work, a compression-based data reuse framework for integral curves is proposed. Individual and groups of curves are compressed and packed together to greatly reduce the storage cost. Meanwhile, we balance the compression ratio, information loss, and decompression cost in the framework. Compression ratio is the major concern of our approach, which makes the integral curves to be handleable and storable. Information loss directly influences the precision of concrete visualization tasks, which is especially crucial for end users. Decompression cost, including time cost and memory cost, is also in our consideration since fast retrieval directly depends on it. The greatly reduced retrieval cost, especially the time cost, is the major advantage we claim for our approach over on-the-fly tracing. Instead, we do not have strict demand on the compression cost, since we expect it to be amortized by future repetitive retrieve quests in a long term. To achieve a reasonable balance among these factors, our approach adopts different compression strategies for seeds blocks of different sizes, which is based on octree space partitioning.

Our method could greatly benefit flow visualization tasks, especially in a limited-resource environment. With preprocessing of tracing and compressing, the compressed integral curves are saved in cheap persistent storage. For flow visualization applications, they could directly retrieve the integral curves using much less time and memory compared to those on-the-fly particle tracing methods. Either in workstations where the volume of flow fields could be larger than the memory, or in clusters where parallel tracing techniques are adopted, our experiments show that the compression-based reuse framework could provide generally tens of times acceleration for large datasets. On the other hand, the underlying flow fields and the tracing process are hidden in our reuse framework. This means our approach could be easily extended to more complex data, such as hybrid-sigma curvilinear mesh, unstructured mesh, etc. For unstructured mesh, particle tracing is much slower than in regular grids, while our compression scheme could still provide fast retrieval regardless of the grid complexity.

The remainder of this paper is organized as follows. We describe the background of our approach in Section 2. In Section 3, we introduce the compression-based reuse framework. We then give more details about parameter settings and parallel implementation in Section 4 and Section 5. Application cases are shown in Section 6 to demonstrate the effectiveness of our framework. In Section 7, we discuss certain open questions in our method. Finally, we conclude this paper with a brief review and discuss the future work.

2 Related Work

In this section, we briefly review some related research, including Lagrangian-based flow visualization, fast integral curve retrieval, and compression techniques in scientific applications.

2.1 Lagrangian-based Flow Visualization

Eulerian specification and Lagrangian specification are two methods to describe unsteady flow data. Both of these two specifications are useful in different scenarios. While the Eulerian perspective is related to instantaneous flow features, time-dependent patterns are usually identified by Lagrangian techniques. One category of Lagrangian-based techniques only requires advection of a set of locally or sparsely placed seeds, which we call local-range analysis, such as streamlines, stream surfaces, streak surfaces [13, 12, 23], etc. Another category, full-range analysis, needs densely place seeds and advecting over the entire domain, such as line integral convolution (LIC) [4], Finite-Time Lyapunov Exponent (FTLE) [17, 11], etc. Besides, densely seeded integral curves are also used to discover flow features according to pathlines attributes [14, 19], or to identify differences of transportation between ensemble flow simulation results [15, 20].

Full-range Lagrangian analysis on flow fields usually requires massive particle tracing in the field, which is computational- and data-expensive. Even for local-range analysis, when the flow fields become larger and more complex, the computation, I/O, and communication cost also increase to a significant scale. Integral curves reuse framework is proposed to overcome the

high retrieval cost caused by data scale and task complexity.

2.2 Fast Integral Curve Retrieval

Particle tracing is the fundamental technique in flow visualization to retrieve integral curves. As the data increases to a large scale, particle tracing becomes an inevitable bottleneck for analyzing large and complex flow fields. Parallel particle tracing is an efficient solution. There are mainly three parallel strategies, i.e. task-parallelism, data-parallelism, and hybrid-parallelism. Task-parallelism mainly focuses on the workload balance across different work nodes [28, 29, 26], while data-parallelism focuses on the data partition [8, 27]. More recently, hybrid-parallelism becomes popular [5, 25]. DStep [22], a MapReduce-like particle tracing framework, becomes one of the most scalable methods. Some flow analysis methods are built on DStep [14, 15] to improve their scalability and efficiency. In this work, we couple integral curve compression with DStep to handle particle tracing for large datasets. Besides parallel computation, access pattern estimation is an effective, orthogonal way to improve performance of particle tracing. Access pattern graph is widely used to optimize data storage [7], to provide sparse data management [16], or to dynamically balance workload [28].

Perhaps the most similar work is the *Hierarchical Line Integration* proposed by Hlawatsch et al. [18], where long integral curves are built from precomputed short ones in a hierarchical way to reduce the computational cost. Both their work and ours are designed to provide fast but lossy integral curve retrieval to avoid expensive particle tracing. The major difference is that we directly start from the uncompressed pathlines, i.e. ground truth, and control the compression error. While their method starts from end points of short integral curves, and performs interpolation on them. Theoretically, our approach could reach any small compression error, or even lossless one. Moreover, our method could easily apply on more complex datasets such as unstructured grid mesh.

2.3 Compression Techniques in Scientific Applications

To visualize or analyze large scale scientific data, compression is one of the most useful techniques. In scientific visualization, compression techniques mainly apply on volume data. In 1994, Tao et al. proposed a wavelet-based compression method [32], which keeps high precision of data by lossless compression. Ratanaworabhan et al. [30] proposed a fast lossless compression method for scientific floating-point data based on a fast hash-table lookup prediction. Fout et al. [10] improved the prediction-based compression method by proposed an adaptive prediction-based approach. To improve the compression ratio, JPEG2000 is one general-purpose method [33] providing both lossless and lossy mode. Another efficient and fast compression method is fpzip method [21], which works well on various kinds of dataset with high throughout. Recently zfp method [24] is proposed as a nearly lossless method, which allows users to specify precision, and enables random block access. To accelerate the compression speed, parallelizing compression methods is also an effective way [3].

However, we could only find one work which uses compression for pathlines [9]. A prediction-based lossless algorithm is used in this work, which reduces the storage of particles by 41% and speeds the particle retrieval by about 20%. While lossy algorithms are proposed in our reuse framework, and the whole problem is considered as a multi-objective optimization problem. We could obtain a much higher compression ratio and much more speedup.

3 Compression-based Integral Curve Reuse Framework

Figure 1 gives an illustration of the flow visualization pipeline which is equipped with our compression-based integral curve reuse framework. Our framework mainly includes two parts: the compression part, and the decompression part. The former part only runs once for one dataset, which conducts numerical integration for dense seeds, then compress the curves, and save the results to persistent storage. It is designed to run on a large cluster to reduce the computation time. After the one-time running of compression part, the decompression part could be deployed on limited-resource workstations to provide fast retrieval of integral curves for visualization applications. Of course, to compete with parallel tracing approaches, parallelized decompression is also support.

3.1 Design Goal

Integral curve compression is the core technique to achieve the goal of data reuse, as the main obstacle comes from the large volume of the intermediate data, i.e. integral curves. We consider it as a problem of multiple objectives:

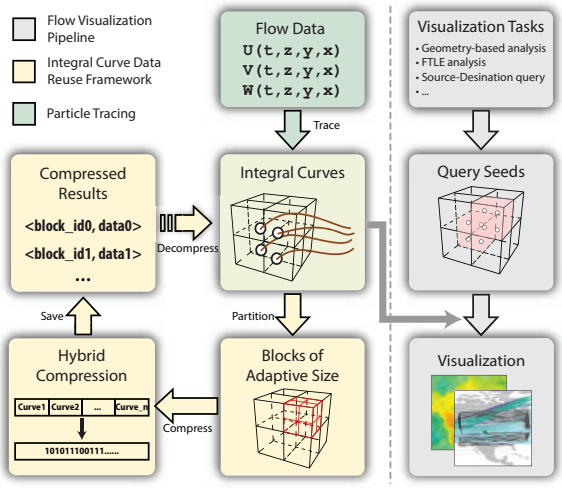

Fig. 1 Flow visualization pipeline equipped with compression-based integral curve reuse framework.

High compression ratio. Particle tracing usually takes hundreds or thousands of integration steps to generate integral curves for seeds, which makes the size of integral curves be hundreds or thousands times larger than the original flow field if we place seeds on every grid point. In our compression scheme, a high compression ratio is highly desired for the storage. Though compression ratio is defined as $\frac{\text{Uncompressed Size}}{\text{Compressed Size}}$, we usually use $\frac{1}{\text{Compression Ratio}}$ for comparison in the following sections.

Controllable information loss. High compression ratio is usually achieved at the price of precision loss of data. Low information loss ensures that we do not take much uncertainty into the visualization results. For various datasets and visualization tasks, end users may have different tolerance on the information loss of integral curves. We would like to provide certain mechanism to help control information loss for different scenarios. The definition of information loss on integral curves is defined in Section 3.2.1.

Fast retrieval. The purpose of our compression-based reuse framework is to replace expensive particle tracing procedure using an efficient integral curve provider. Fast retrieval is the key factor to accelerate visualization applications. The greatly reduced retrieval cost is also the major contribution we claim for our reuse framework.

However, compression ratio is an objective that opposes to the other two, which is the common situation in data compression problems. Trade-off among these objectives is required in our compression scheme. On one hand, high compression ratio is usually obtained by lossy compression methods by lowering data precision. We have applied lossless gzip algorithm to compress a bunch of integral curves, which could only obtain a compression ratio of about 1.43. Even with prediction step to decrease the entropy of data, the lossless compress could only achieve compression ratio of 1.59 [9]. Instead, certain lossy float-point compression methods, like fpzip or zfp, could easily reach 5.0 or even higher ratio, yet also bring information loss. On the other hand, to squeeze more redundant information from integral curves also indicates that more time is needed to restore the information. For 4D (time included) integral curves, spatiotemporal coherence is the major source of redundancy. For some compression methods, compressing more coherent integral curves at once is expected to improve the compression ratio, but the decompression cost is also increased.

To balance the three objectives, a hierarchical and hybrid compression scheme is designed. Specifically, individual curve compression method and curve group compression method are combined to achieve high compression ratio as well as to keep low information loss. In our current implementation, we choose Bézier curve fitting and fpzip method respectively. The hybrid compression methods are applied to blocks of adaptive sizes on the octree partitioning to balance the decompression cost and compression ratio. As a general framework, it is viable to replace the specific compression algorithms with other ones, only if they could satisfy previous goals. We also want to try more algorithms in the future.

3.2 Hybrid Compression on Integral Curves

Lots of compression methods have been proposed, yet not all of them are suitable for integral curve compression. General lossless algorithms usually do not achieve high compression ratio for integral curve compression. In our work, we mainly investigate two kinds of data-specific compression methods, which are able to obtain much better results. One kind of the methods is specific to compress individual curve by using a sparse representation of the digitalized curve, where we choose *Bézier curve fitting*. The other kind is designed to efficiently compress curve groups as a 3-D scalar field, where *fpzip library* is chosen. To further improve compression ratio and to lower the error, hybrid usage of compression methods is also used in our compression scheme. We should note that we choose these specific compression techniques and combine them, only because they are ready for use and have achieved satisfying results in our experiments. Lots of other techniques could be the replacements as long as they could

achieve good compression ratio and keep controllable error.

In the following, we first formalize error measurement in integral curve compression, and then we introduce two representatives of lossy compression methods, Bézier curve fitting and fpzip library, and their hybrid usage.

3.2.1 Error Measurement

Flow map formally describes integral lines, including pathlines and streamlines. It is an important concept in Lagrangian-based analysis, which is a function of spatiotemporal location $\mathbf{x}_0^{t_0}$ and time t:

$$\Phi : (\mathbf{x}_0^{t_0}; t) \mapsto \Phi_{t_0}^{t_0+t}(\mathbf{x}) = \mathbf{x}(\mathbf{x}_0^{t_0}; t),$$

where $\mathbf{x}(\mathbf{x}_0^{t_0}; t)$ indicates the spatiotemporal location of seed $\mathbf{x}_0$ after advection of time t. In the following, we use $\mathbf{x}(t)$ to denote $\mathbf{x}(\mathbf{x}_0^{t_0}; t)$ for concise presentation.

Now we describe the discrete case of integral curves compression. The integral curve seeding from $(\mathbf{x_0}, t_0)$ is denoted as a sequence of points:

$$\mathbf{x}(t_0), \mathbf{x}(t_1), \cdots, \mathbf{x}(t_{m-1}),$$

where m is the number of sample points. After applying certain compression algorithm, the curve is transformed into

$$\mathbf{x}'(t'_0), \mathbf{x}'(t'_1), \cdots, \mathbf{x}'(t'_{m-1}).$$

We should note, in the compression, not only the curve may be distorted ($\mathbf{x} \to \mathbf{x}'$), the advection time of each sample points may also be different ($t \to t'$).

To measure the error introduced by compression, we actually compute the difference between $\mathbf{x}$ and $\mathbf{x}'$. However, in the discrete form of the integral curve, since we do not have information other than those sample points, we choose to measure the displacements of sample points instead of the whole curve. Moreover, we prefer using the maximum displacement of sample points than averaged, because we do not want those large displacements to be amortized. Therefore, the error of compression on one integral curve is defined as

$$\max_{0 \le i < m} ||\mathbf{x}(t_i) - \mathbf{x}'(t_i)||,$$

and the unit is voxel. For certain applications, one may want to use meter or kilometer to measure the displacement. But in this work, we just choose voxel for the demonstration of our method.

For all seeds in the datasets, we use the maximal error D to measure the overall quality of compression. To summarize, the goal of hybrid compression is to control the overall compression error D as well as to reach high compression ratio.

3.2.2 Bézier Curve Fitting

Bézier curve fitting is one of those compression methods which are specific to compress individual integral curve. It calculates the sparse representation of digitalized curves. We choose Bézier curve mainly because its simplicity compared to B-splines or NURBS, especially in the aspect of curve fitting.

A Bézier curve is a frequently used parametric curve in computer graphics and related fields. Schneider has explored the possibility of fitting digitalized curves using Bézier curve [31]. This adaptive algorithm automatically fits a piecewise cubic Bézier curve to a digitized curve in a geometrically continuous (G^1) way. For details of this approach, please refer to the original work. When we adopt Bézier curve fitting algorithm for the purpose of compression, we treat each integral curve $\mathbf{x}(\mathbf{x}_0^{t_0})$ as a sequence of points, and apply the fitting algorithm. If we need n Bézier segments to represent original curve with length m, the compression ratio is $\frac{m}{n*3+1}$ (neighboring segments share one control point). In best cases, only 1 Bézier segment (4 control points) is sufficient to represent original curve, so the compression ratio would be $\frac{m}{4}$.

Bézier curve fitting is especially good at compressing individual integral curve with moderate `error_bound`. Usually, most curves could be compressed to 8% of original size with displacement smaller than 0.5 voxel, which is outstanding in our experiments. Moreover, Bézier curve fitting has one significant advantage that the output of the compression has the same format (list of points) with the original curves. Such loose organization of data still has the potential for further compression. However, with a strict `error_bound` is used, there is a possibility the "compressed" curve is larger than the original one. Thus Bézier curve fitting may not work well for those high-precision visualization applications.

3.2.3 Fpzip Library

Different with Bézier curve fitting that works on individual curves, another category of algorithms is able to compress a bunch of curves efficiently. Fpzip, proposed by Isenburg and Lindstrom [21], is a library for lossless or lossy compression, which has been successfully applied on floating-point scalar fields. This algorithm can be used to compress various kinds of datasets with variable-precision floating-point and integer data, such as unstructured meshes, point sets, images, and voxel grids. In our compression scheme, we treat groups of 3-D integral curves as 4-D floating-point array to apply fpzip library. In the future, we

could investigate the behavior of more similar compression techniques, such as wavelet transform.

Fpzip library provides the parameter `precision` to control the error. It specifies the number of bits in the floating-point number to b used for compression. When applying fpzip library to integral curve compression, we need to deal with two problems. The first one is to pair the dimensions of integral curves with the ones used in the fpzip library, so that high spatiotemporal coherence is kept. Because fpzip library only accepts 3-D floating-point array, we need to split the 4-D array into multiple 3-D ones along one dimension. In our experiment, we found splitting along dimension z could better exploit the coherence to obtain better compression ratio. The other problem is how to deal with the different lengths of integral curves. Since variable-length arrays are not accepted by the fpzip library, we pad each curve with special values, e.g. `NaN`, to the same length. The experiment has shown that fpzip library is efficient to handle such padding with almost no overhead.

Fpzip library provides not only lossy compression but also lossless one with relatively high compression ratio. In case we need to keep very small or no error (less than 0.1 voxels), fpzip could achieve better compression ratio than Bézier curve fitting. The merit of the fpzip library could help our compression scheme to satisfy wider requirements of visualization tasks.

3.2.4 Hybrid Compression

Our experiments (see Section 4) have shown that using one method, Bézier curve fitting or fpzip, separately could over-dominate those general-purpose lossless compress algorithms in the aspect of compression ratio. Yet both methods also have their own shortcomings. Bézier curve fitting usually performs badly when very small error needed, while fpzip need more data to keep information loss of medium level. We propose to combine Bézier curve fitting and fpzip method expecting they complement each other.

The inspiration of such combination comes from the property of Bézier curve fitting, that is its output, control points, still have the same format as original curves. It means that compression methods which could apply to the original curves can still apply to these control points. Moreover, for neighboring curves, they are not only spatially coherent before compression. Even after Bézier curve fitting, there still exists spatial coherence among the control points. Based on the observation, we propose hybrid compression to apply fpzip library on the results of Bézier curve fitting. Our experiments also successfully demonstrate the efficiency of the hybrid compression.

Right now, with the hybrid usage of Bézier curve fitting and fpzip library, our compression scheme is already able to satisfy various requirements, such as lossless compression, or high compression ratio over 50, etc. Our framework is still open to more compression methods for improvements.

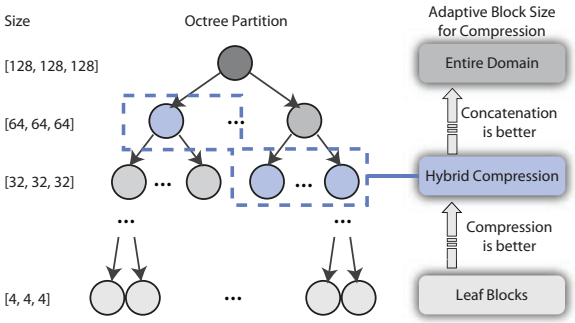

Fig. 2 Illustration of compression on blocks of adaptive sizes based on the octree space partitioning. There are two choices to compress the integral curves for every node (block): either directly apply hybrid compression, or concatenate the compressed results of its 8 children nodes (blocks). After making choices for every node, we actually compress several blocks of different sizes (blue-colored nodes).

3.3 Octree Space Partitioning

The hybrid compression mentioned before only considers the trade-off between compression ratio and information loss. While, for certain methods such as the fpzip library, both compression ratio and decompression cost are highly related with the block size. To balance compression ratio and decompression cost, we adopt octree structure to hierarchically partition the domain into adaptive block sizes for hybrid compression.

Octree space partitioning is a common technique to represent objects in 3D space. A common usage of octree space partitioning is to choose suitable level-of-details in 3D space based on a certain measure. In our framework, we adaptively select better block sizes for compression along the octree structure (Figure 2). To be specific, for one node in octree structure, e.g. one seeding block, we have two ways to compress its corresponding integral curves. One way is to directly apply hybrid compression, which is expected to fully take advantage of the spatial coherence of all curves to obtain high compression ratio. The other one is to just concatenate the compressed results of its 8 children nodes (sub-blocks) as the compressed result of the current block. This approach would avoid decompressing

massive integral curves at one time, which is especially useful when we only need sparse ones. After making choices for every node, we actually compress several blocks of different sizes (blue-colored nodes in Figure 2)). For these blocks, we store their identities and corresponding compressed results in persistent storage for future usage.

It is not easy to choose between these two ways, because we cannot accurately predict the decompress time when doing compression. To help make choices, we first we define an index ρ to measure the benefit brought by direct hybrid compression to one block, i.e. $\rho = \frac{\text{direct compression size}}{\text{size of concatenation}}$. Smaller ρ indicates that direct hybrid compression brings more significant improvement in compression ratio, thus direct compression is more preferred for this block. Otherwise, concatenation is a better choice. Users are allowed to set a `threshold` for ρ to control the compression strategies of each block. Based on the setting `threshold`, we are able to make decisions for all octree nodes to obtain the adaptive size scheme. If the ρ of one block is smaller than the `threshold`, direct hybrid compression is applied, otherwise concatenation. End users could set the threshold based on their desired decompression cost.

However, to obtain a reasonable threshold of ρ is still tricky, because decompression cost is unknown before compression and is also sensitive to many factors, such as different datasets, regions, etc. We have conducted several experiments on different `thresholds` and measure the total compression ratio and the total decompression cost. Then we empirically pick up one `threshold` which leads to a good balance of the two factors.

3.4 Decompression Utility

After efficiently compressing and storing integral curves, the decompression utility is expected to provide fast retrieval for visualization applications. The general decompression routine is straightforward: first locating the blocks that contain the seeds, then decompressing stored data and collecting necessary curves. Previous compression scheme has stored results in block-data pairs. When end users request the integral curves of a set of seeds in one region, those blocks intersected with the query block are picked out. With octree structure, these blocks could be quickly located. Then their corresponding compressed data are decompressed, and the requested integral curves are collected and returned for visualization applications.

We make more efforts to improve the time efficiency and memory usage of decompression utility. On one hand, if the decompression utility is run for just one-time use, the decompressed data would be discarded after the usage, leading to a waste of computation resources. In fact, we let the decompression utility act as a long-term running server to provide integral curves as requested. In this situation, the decompressed subsets of integral curves could be cached in memory to accelerate the following retrieval. On the other hand, because the decompression routine for each block-data pairs is embarrassing parallel, the decompression server could be easily deployed with parallel acceleration. Various parallel techniques, such as OpenMP or MPI, can work well to exploit different amount of resources. Our experiments (see Section 6) have shown that the decompression time cost is much less than on-the-fly particle tracing.

4 Configurations of Compression Scheme

Compression-based integral curve reuse is treated as a problem of multiple objectives, which depends on several variables. Compression methods and their parameters are major factors to reach high compression ratio and to control error. At the same time, the `threshold` of ρ is used to balance the compression ratio and decompression cost. We have conducted several experiments to study the impact of different configurations to the three objectives. These results not only describe the relationships between these variables and the compression results, but also give guidance to end users to help choose configurations for their concrete visualization.

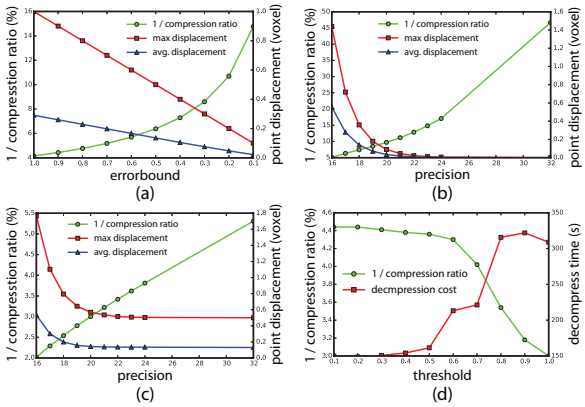

Fig. 3 (a)-(c) shows the compression ratio and point displacement under different compression methods and parameters: Bézier curve fitting, fpzip algorithm, and hybrid compression respectively. (d) shows the compression ratio and decompression cost under different `thresholds` of ρ using hybrid compression.

4.1 Parameters of Compression Methods

In our compression scheme, compression methods and their parameters are the main factors which influence the balance between the compression ratio and information loss. We have conducted several experiments to study their influence to compression ratio and information loss. The statistics of experiments are shown in Figure 3. Three compression methods, (a) Bézier curve fitting, (b) fpzip algorithm, and (c) hybrid usage of Bézier curve fitting and fpzip algorithm with `error_bound` set to 0.5, are tested using Hurricane Isabel dataset. We could obtain some useful guidance from the charts:

- Bézier curve fitting is very suitable in scenarios which need strictly control error, because its maximal point displacements are very close to the set `error_bound`.
- Fpzip algorithm is good at keeping low error when `prec` is set larger than 22. It can even achieve lossless compression with reasonable compression ratio.
- When high compression ratio is urgently needed, combining Bézier curve fitting with the fpzip algorithm is a good choice. By setting `prec` larger than 22, the maximal point displacement almost keeps the same of the set `error_bound`, but the compressed data size is almost halved.

These observations also confirm the strength and shortcoming we have mentions in the previous section.

Based on the statistics, we could choose some good parameters. We set `error_bound` to 0.5 for Bézier curve fitting, and set `prec` to 19 for fpzip library. For hybrid compression of these two methods, we set `error_bound` to 0.5, and `prec` to 20. With these settings, the maximal point displacement could be kept around 0.5 voxels, and the compression ratio could reach beyond 10 or even more.

4.2 `Threshold` of ρ

In the compression scheme, the `threshold` of ρ is used to control block size adaptively for compression. Different values of the `threshold` are tested to investigate its impact on compression ratio and decompression time. While other compression parameters are set as the values we obtain previously. In the experiments, pathlines of densely placed seeds are retrieved.

Figure 3(d) shows the compressed data size and decompression time under different values of `threshold` in Hurricane Isabel dataset. As `threshold` decreases from 1.0 to 0.1, the compression ratio also decreases. And the decompression time also has the same trend. We could observe a jump of decompression time around the value 0.7 and 0.5, which indicates that there exist lots of blocks which have ρ of these values. Instead, the compression ratio changes smoothly at these values. Thus, to set `threshold` to these jumping points could bring lots of benefit of decompression time with little punishment on compression ratio. For example, we could set the `threshold` to 0.7 for Hurricane Isabel dataset.

5 Parallel Implementation

Though the time cost of the tracing and compressing part is not the major concern of our reuse framework, we still expect it not take too much time. Parallel particle tracing could be one good solution for this purpose. On the other hand, to avoid huge and unnecessary disk read/write operations of uncompressed pathlines, the *"in situ"* way of integral curve compression could be better than post-processing. Therefore, we propose to incorporate our integral curve compression with one of the most scalable particle tracing solution, DStep [22]. DStep is a MapReduce-like framework designed for domain traversal, where field line tracing is one of its direct applications. Developers only need to implement `step()` (similar to `map()`) and `reduce()` functions with key-value pairs for domain traversal-related tasks, without need to manually manage communication and job scheduling in parallel environment. For massive particle tracing, partial pathlines are traced from seeds in the step stage, and then merge into complete pathlines in the reduce stage. To embed the compression scheme into the DStep framework, we use the seeding blocks of seeds as their keys. Then integral curves from the same seeding blocks are collected to conduct octree space partitioning and the following hybrid compression.

We deploy our parallel compression system on our cluster. We test compression on Hurricane Isabel dataset (a spatial domain of $500 \times 500 \times 100$, 48 timesteps) with up to 64 processes. It takes 10,383s for the hybrid compression to compress 25M pathlines seeding from the first timestep. Although the compression time is significantly larger than particle tracing for this dataset, but it only needs to run once. The time cost could be amortized by future usage in a long term. For example, if we want to calculate FTLE field in one single workstation whose memory is smaller than 12GB, we could save about 1,000s each time (see next Section).

#cores	2	4	8	16	32
(a)	91.27	48.35	27.87	17.63	12.38
(b)	92.85	51.85	37.41	44.06	44.16
(c)	7610.83	1468.21	959.54	1023.66	874.70

Table 1 Elapsed time of pathline retrieval for Hurricane Isabel data: (a) our method, (b) in-core naive particle tracing, and (c) out-of-core LDB particle tracing.

6 Case Study

We demonstrate the effectiveness of our reuse framework by comparing our approach with on-the-fly particle tracing. Three datasets are considered in our case study, i.e. Hurricane Isabel dataset, Ocean Simulation dataset, and Large Eddy Simulation dataset. Depending on the properties of datasets, such as data size and grid mesh types, and specific visualization tasks, the baseline methods to compare could vary. We intend to cover application scenarios as broadly as possible to show the benefits of our approach, especially in a resource-limited environment.

6.1 Hurricane Isabel Dataset

In this case, we compute FTLE field on Hurricane Isabel dataset to demonstrate the effectiveness of dense integral curve retrieval for datasets of medium or small sizes. Hurricane Isabel dataset is from an atmospheric simulation, whose spatial resolution is $500 \times 500 \times 100$. It simulates the advance of one hurricane in a physical scale of 2,139km$\times$2,004km$\times$19.8km. This data contains 48 timesteps, which are saved per hour in the simulation. Three speed variables, U, V, and W, are extracted for our computation, which results in $\sim$14GB data. Right now, in our reuse framework, we only consider place seeds at all grid points of the first timestep, which results in totally $\sim$160.25GB uncompressed pathlines. Our approach could compress them into $\sim$8.03GB with a compression ratio of 19.97.

To calculate the FTLE field, we place seeds at the striped grids of $125 \times 125 \times 25$. Thus, about 390k pathlines need to be traced or retrieved. We compare our approach with two particle tracing implementation. One is the out-of-core LDB method [16], which is designed for workstations with limited memory. The other is the in-core naive method which reads all data into memory at once and traces seeds using multi-threads, which is more suitable when memory is sufficient. Table 1 shows the elapsed time of these three methods under different numbers of cores. We could observe that our compression approach provides very fast pathline retrieval compared to out-of-core particle tracing. Compared to the in-core method, our reuse framework still obtains comparable retrieval efficiency when #cores is small, or even better when #cores is large.

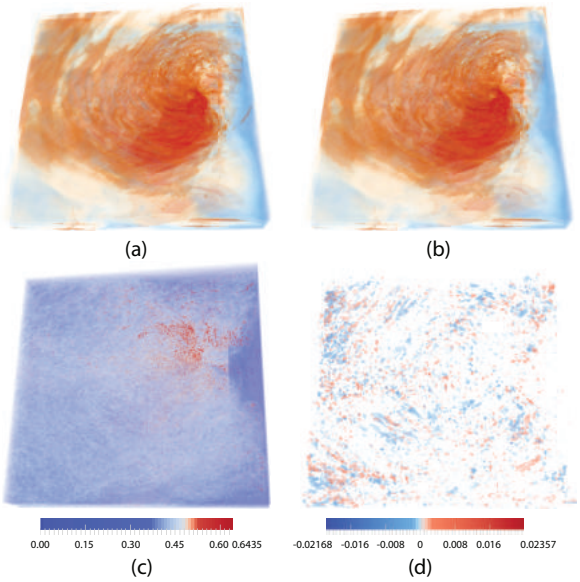

Fig. 4 Rendering results of (a) FTLE field using particle tracing, (b) FTLE field calculated using our reuse framework with parameters `error_bound=0.5, prec=19, threshold=0.7`, (c) compression error field of pathlines, and (d) the FTLE difference field.

Figure 4 shows the visualization results of FTLE field computation from (a) uncompressed pathlines and (b) our compression-based reuse framework with parameters `error_bound=0.5, prec=19, threshold=0.7`. The rendering results of these two methods are almost the same. We further render the differences of pathlines seeding from every grid point in (c) and the difference of FTLE field in (d). We could observe that most pathlines have the compression error near or less than 0.5, but some positions near the hurricane eye have a large error of more than 0.6, which might be related to the high wind speed in that region. This fact also inspires us to adapt different parameter configurations for different regions in one dataset. FTLE different field shows that the differences of FTLE values range from -0.02168 to 0.02357, about 5% of relative error. But actually, most regions have the difference values between -0.002 and 0.002, which is set transparent in the figure.

6.2 Ocean Simulation Dataset

This dataset comes from ocean simulation. It has a very high spatial resolution of 3603$\times$1683, and 55 depth levels. The temporal resolution is 1 day. We used 20-day of data for our experiment which has the size of

#cores	2	4	8	16	32
(a)	27.67	19.90	16.24	15.99	12.58
(b)	488.092	503.96	466.37	462.74	634.40

Table 2 Elapsed time of pathline retrieval for Ocean Simulation data: (a) our method, and (b) out-of-core LDB particle tracing.

∼72.2GB. Seeds are placed at every grid point of the first timestep, and traced at most 512 steps. The total volume of uncompressed pathlines is ∼827GB, and is compressed to ∼93GB. The compression ratio is ∼8.89. In this case, we retrieve 135K pathlines (seeding in the gray box in Figure 5(a)), and render them.

The elapsed time of our reuse framework and particle tracing is shown in Table 2. Our approach could accelerate the pathline retrieval process from 18 to 52 times as the number of cores increase. And those in-core particle tracing methods even could not work in this resource-limited environment. The Figure 5 shows the rendering result of pathlines (a), and the compression error field of pathlines (b). The maximal difference could reach 1.796, but is still tolerable under such high spatial resolution. We could also observe some vortex-like structures in this error field, tagged using green boxes. These structures could be related with the Lagrangian Coherent Structures (LCS), which need further study.

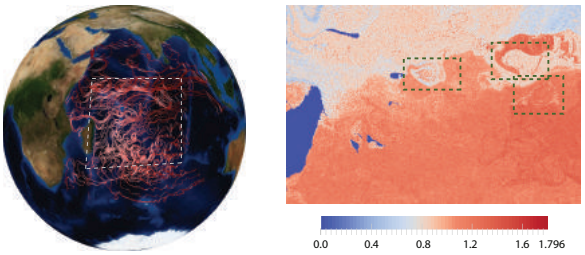

Fig. 5 Rendering results of (a) pathlines using our approach with parameters `error_bound=0.5, prec=19, threshold=0.7`, (b) compression error field of pathlines.

6.3 Large Eddy Simulation Data

When the grid complexity of flow fields increases, it becomes more expensive and uneconomical to (re)compute the integral curves. The point locating problems and sophisticated interpolation schemes in unstructured grids lead to much higher computation cost. Our framework can deal with these problems, since it is a mesh-independent data access method. In this case, we test a result of large eddy simulation (LES), where flow passes a ball at supercritical regime ($R_e = 1.14e6$). The simulation data contains totally ∼181M cells constructed by ∼67.8M nodes, which has the size of ∼11.75GB (Figure 6(a)). We placed (25^3 =)15,625 seeds around the ball and traced pathlines with at most 512 steps. It took about 70 minutes for ParaView in a single workstation to extract these streamlines, whose size is ∼41.26MB. The original streamlines are rendered in Figure 6(c).

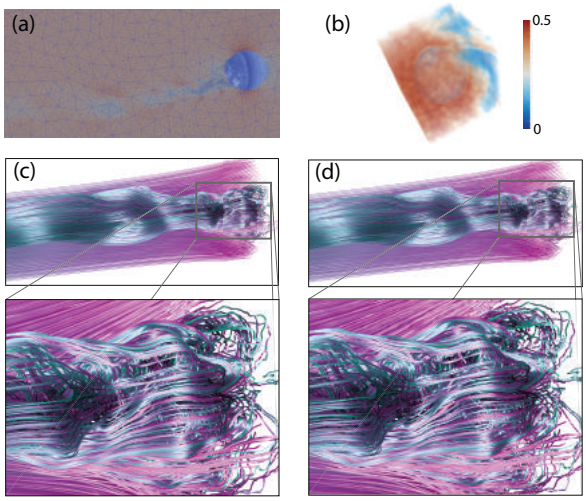

Fig. 6 Rendering results of (a) Original triangular meshes, (b) Compress error field of streamlines, (c) Streamlines obtained by particle tracing, and (d) Streamlines obtained from hybrid compression with `error_bound=0.5, prec=24`.

These streamlines are passed to our compression scheme. Hybrid methods with `error_bound=0.5, prec=24` is used to compress them, which results in ∼652.84KB of compressed data. To retrieve these streamlines, the decompression utility only costs 5.40s, which is significantly faster than particle tracing. The rendering result is shown in Figure 6(d), which is very similar to the results from particle tracing. The compression error field is rendered in (b), where the ball region is set transparent. We could observe the streamlines seeding from the right side of the ball have smaller compression error, while other positions have larger ones.

In a summary, the three cases above have successfully demonstrated the major benefits of our compression-based reuse framework, that is to greatly reduce the retrieval cost of integral curves, to handle large-scale flow visualization in a resource-limited environment, and to overcome the complexity and variety of flow data.

7 Discussions

The previous case study has successfully demonstrated that our compression-based integral curve reuse framework enables fast integral curve retrieval on limited-resource machines, either for datasets of various sizes, or of different types of grid mesh. However, there still exist certain open questions with our current implementation.

Choices of Compression Algorithms and Parameters. In our reuse framework, we need to solve a multi-objective optimization problem, which makes trade-off among compression ratio, information loss, and decompression cost. The variables we could control are compression algorithms, parameters for each algorithm, and `threshold` of ρ. None of them is easy to make a decision. The main reason is the difficulty to exhaust every choice of the variables. In our current experiments, we usually fix certain parameters and study the influence of the others. There could be a very time-consuming work to exhaust all possible compression algorithms and their following parameters. One possible solution is to analyze certain properties of the data, e.g. entropy, FTLE, etc., and then choose parameters based on their properties. Moreover, how one configuration of parameters works on general datasets still needs in-depth study. In our case study, we directly apply the configuration of Hurricane Isabel to Ocean Simulation dataset. The compression ratio becomes a little lower, but it is still a huge improvement compared to lossless compression algorithms. The compression error is larger, but still tolerable considering the high spatial resolution. At the same time, the retrieval efficiency is still great compared to on-the-fly particle tracing.

Influence of Compression to Visualization Tasks. Error control is one important task for lossy compression algorithms. In our approach, we could measure the compression error for each integral curve, and then control the error by setting suitable parameters. However, how these errors of integral curves influence visualization tasks needs thorough study. It is also the situation of some other fast integral curve retrieval works [18, 1]. Although we could analyze the accuracy of each integral curve, it keeps difficult to estimate their influence on further visualization tasks, such as LCS extraction, source-destination query, etc. One possible go-around solution for such questions is to estimate the error bound of our compression scheme and the accumulated error of numerical integration in particle tracing, and then to show the error brought by compression is no larger than numerical integral under certain compression parameter settings. We also note that some other works directly approximate the flow map or FTLE field using fewer samples [11, 2]. However, these methods are more specific compared to our integral curve reuse framework, since our approach is designed for general integral curve-based flow visualization tasks.

Compression of Integral Curves from Different Seeding Timesteps. In the current implementation, integral curves from different seeding timesteps are compressed independently. We do not consider the coherence of integral curves from neighboring seeding timesteps right now. One major concern is that most visualization tasks, e.g. FTLE computation, LIC, etc., prefer retrieving integral curves from the same seeding timestep other than from different timesteps. It could increase the decompress cost if we compress them together but only retrieve curves from one seeding timestep. At the same time, the coherence from neighboring seeding timesteps depends on the temporal resolution largely, while the spatial coherence is widely acknowledged. However, this idea is still worth trying with suitable datasets and application.

8 Conclusions and Future Work

In this paper, we propose a novel approach to take advantage of reusability of integral curves to achieve very fast integral curve retrieval in a resource-limited environment. Our method is designed to provide fast retrieval as well as to keep storage efficient and information loss low.

In the future, we could improve our framework from the following directions. First, as for the compression part, we want to test more compression algorithms in current scheme to further improve our objectives. At the same time, we would like to provide a data-oriented, automatic, and time-efficient mechanism for parameter selection. It aims to decide appropriate parameters for data blocks based on their properties. Second, it is possible to augment the current framework with integral line interpolation techniques. The current implementation requires the query seeds of visualization tasks to be the grid points of the original data. If some tasks need streamlines or pathlines seeding from non-grid points, we have to re-run the compression part in a finer seeding grid. By introducing interpolation techniques [18, 6], such re-computation is avoided, and the application scenarios of our framework could be expanded. Third, in the theory part, analytic estimation of the compression error with respect to the compression and sampling parameters would offer great help for users to choose the parameters. It is also in demand

to study the propagation of the error to the concrete visualization tasks.

Acknowledgements This work is supported by NSFC No. 61672055. This work is also partially supported by NSFC Key Project No. 61232012 and the Strategic Priority Research Program - Climate Change: Carbon Budget and Relevant Issues of the Chinese Academy of Sciences Grant No. XDA05040205.

References

1. A. Agranovsky, D. Camp, C. Garth, E. W. Bethel, K. I. Joy, and H. Childs. Improved post hoc flow analysis via lagrangian representations. In *Proc. of IEEE LDAV Symposium*, pages 67–75, 2014.
2. S. S. Barakat and X. Tricoche. Adaptive refinement of the flow map using sparse samples. *IEEE Trans. Vis. Comput. Graph.*, 19(12):2753–2762, 2013.
3. C. Bi and K. Ono. 2-3-4 combination for parallel compression on the K computer. In *Proc. of IEEE Pacific Visualization Symposium*, pages 281–285, 2014.
4. B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Proc. of the 20th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 263–270, 1993.
5. D. Camp, C. Garth, H. Childs, D. Pugmire, and K. I. Joy. Parallel stream surface computation for large data sets. In *Proc. of IEEE LDAV Symposium*, pages 39–47, 2012.
6. J. Chandler, H. Obermaier, and K. I. Joy. Interpolation-based pathline tracing in particle-based flow visualization. *IEEE Trans. Vis. Comput. Graph.*, 21(1):68–80, 2015.
7. C. Chen, L. Xu, T. Lee, and H. Shen. A flow-guided file layout for out-of-core streamline computation. In *Proc. of IEEE Pacific Visualization Symposium*, pages 145–152, 2012.
8. L. Chen and I. Fujishiro. Optimizing parallel performance of streamline visualization for large distributed flow datasets. In *Proc. of IEEE Pacific Visualization Symposium*, pages 87–94, 2008.
9. D. Ellsworth, B. Green, and P. J. Moran. Interactive terascale particle visualization. In *15th IEEE Visualization 2004 Conference (VIS 2004)*, pages 353–360, 2004.
10. N. Fout and K.-L. Ma. An adaptive prediction-based approach to lossless compression of floating-point volume data. *IEEE Trans. Vis. Comput. Graph.*, 18(12):2295–2304, 2012.
11. C. Garth, F. Gerhardt, X. Tricoche, and H. Hagen. Efficient computation and visualization of coherent structures in fluid flow applications. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1464–1471, 2007.
12. C. Garth, H. Krishnan, X. Tricoche, T. Tricoche, and K. I. Joy. Generation of accurate integral surfaces in time-dependent vector fields. *IEEE Trans. Vis. Comput. Graph.*, 14(6):1404–1411, 2008.
13. C. Garth, X. Tricoche, T. Salzbrunn, T. Bobach, and G. Scheuermann. Surface techniques for vortex visualization. In *Proc. of Symposium on Visualization*, pages 155–164, 346, 2004.
14. H. Guo, F. Hong, Q. Shu, J. Zhang, J. Huang, and X. Yuan. Scalable Lagrangian-based attribute space projection for multivariate unsteady flow data. In *Proc. of IEEE Pacific Visualization Symposium*, pages 33–40, 2014.
15. H. Guo, X. Yuan, J. Huang, and X. Zhu. Coupled ensemble flow line advection and analysis. *IEEE Trans. Vis. Comput. Graph.*, 19(12):2733–2742, 2013.
16. H. Guo, J. Zhang, R. Liu, L. Liu, X. Yuan, J. Huang, X. Meng, and J. Pan. Advection-based sparse data management for visualizing unsteady flow. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2555–2564, 2014.
17. G. Haller. Distinguished material surfaces and coherent structures in three-dimensional fluid flows. *Physica D: Nonlinear Phenomena*, 149(4):248–277, 2001.
18. M. Hlawatsch, F. Sadlo, and D. Weiskopf. Hierarchical line integration. *IEEE Trans. Vis. Comput. Graph.*, 17(8):1148–1163, 2011.
19. F. Hong, C. Lai, H. Guo, E. Shen, X. Yuan, and S. Li. FLDA: latent dirichlet allocation based unsteady flow analysis. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2545–2554, 2014.
20. M. Hummel, H. Obermaier, C. Garth, and K. I. Joy. Comparative visual analysis of lagrangian transport in CFD ensembles. *IEEE Trans. Vis. Comput. Graph.*, 19(12):2743–2752, 2013.
21. M. Isenburg, P. Lindstrom, and J. Snoeyink. Lossless compression of predicted floating-point geometry. *Computer-Aided Design*, 37(8):869–877, 2005.
22. W. Kendall, J. Wang, M. Allen, T. Peterka, J. Huang, and D. Erickson. Simplified parallel domain traversal. In *Proc. of International Conference on High Performance Computing Networking, Storage and Analysis*, pages 10:1–10:11, 2011.

23. H. Krishnan, C. Garth, and K. I. Joy. Time and streak surfaces for flow visualization in large time-varying data sets. *IEEE Trans. Vis. Comput. Graph.*, 15(6):1267–1274, 2009.
24. P. Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2674–2683, 2014.
25. K. Lu, H. Shen, and T. Peterka. Scalable computation of stream surfaces on large scale vector fields. In *Proc. of International Conference on High Performance Computing Networking, Storage and Analysis*, pages 1008–1019, 2014.
26. C. Müller, D. Camp, B. Hentschel, and C. Garth. Distributed parallel particle advection using work requesting. In *Proc. of IEEE LDAV Symposium*, pages 1–6, 2013.
27. B. Nouanesengsy, T. Lee, K. Lu, H. Shen, and T. Peterka. Parallel particle advection and FTLE computation for time-varying flow fields. In *Proc. of International Conference on High Performance Computing Networking, Storage and Analysis*, pages 61:1–61:11, 2012.
28. B. Nouanesengsy, T. Lee, and H. Shen. Load-balanced parallel streamline generation on large scale vector fields. *IEEE Trans. Vis. Comput. Graph.*, 17(12):1785–1794, 2011.
29. D. Pugmire, H. Childs, C. Garth, S. Ahern, and G. H. Weber. Scalable computation of streamlines on very large datasets. In *Proc. of the ACM/IEEE Conference on High Performance Computing*, 2009.
30. P. Ratanaworabhan, J. Ke, and M. Burtscher. Fast lossless compression of scientific floating-point data. In *Proc. of Data Compression Conference*, pages 133–142, 2006.
31. P. J. Schneider. An algorithm for automatically fitting digitized curves. In *Graphics gems*, pages 612–626. Academic Press Professional, Inc., 1990.
32. H. Tao and R. J. Moorhead. Progressive transmission of scientific data using biorthogonal wavelet transform. In *Proc. of IEEE Conference on Visualization*, pages 93–99, 1994.
33. B. E. Usevitch. JPEG2000 compliant lossless coding of floating point data. In *Proc. of Data Compression Conference*, pages 484–484, 2005.