Analyzing the Effect of Lossy Compression on Particle Traces in Turbulent Vector Fields

Marc Treib, Kai Bürger, Jun Wu, Rüdiger Westermann
Technische Universität München, Munich, Germany
{*treib,buerger,jun.wu,westermann*}*@tum.de*

Keywords: Vector fields, turbulence, particle tracing, data compression, data streaming.

Abstract: We shed light on the accuracy of particle trajectories in turbulent vector fields when lossy data compression is used. So far, data compression has been considered rather hesitantly due to supposed accuracy issues. Motivated by the observation that particle traces are always afflicted with inaccuracy, we quantitatively analyze the additional inaccuracies caused by lossy compression. In some experiments we confirm that the compression has only minor impact on the accuracy of the trajectories. Even though our experiments are not generally valid, they indicate that a more thorough analysis of the error in particle integration due to compression is necessary, and that in some cases lossy compression is valid and can significantly reduce performance limitations due to memory and communication bandwidth.

1 INTRODUCTION

One of the most intriguing and yet to be fully understood aspects in turbulence research is the statistics of Lagrangian fluid particles transported by a fully developed turbulent flow. Here, a fluid particle is considered a point moving with the local velocity of the fluid continuum. The analysis of Lagrangian statistics is usually performed numerically by following the time trajectories of fluid particles in numerically simulated turbulent fields. Let $\mathbf{x}(\mathbf{y},t)$ and $\mathbf{u}(\mathbf{y},t)$ denote the position and velocity at time t of a fluid particle originating at position $\mathbf{y}$ at time $t = 0$. The equation of motion of the particle is

$$\frac{\partial \mathbf{x}(\mathbf{y},t)}{\partial t} = \mathbf{u}(\mathbf{y},t),$$

subject to the initial condition

$$\mathbf{x}(\mathbf{y},0) = \mathbf{y}.$$

The Lagrangian velocity $\mathbf{u}(\mathbf{y},t)$ is related to the Eulerian velocity $\mathbf{u}^+(\mathbf{y},t)$ via $\mathbf{u}(\mathbf{y},t) = \mathbf{u}^+(\mathbf{x}(\mathbf{y},t),t)$. By using a numerical integration scheme, the trajectory of a particle released into the flow can now be approximated.

Particle tracing in discrete velocity fields of a sufficient spatial and temporal resolution to resolve the higher wavenumber components in turbulent flows is nonetheless difficult. For reasonably-sized particle ensembles, due to the massive amount of data to be accessed during particle tracing, the performance is limited by the available memory bandwidth capacities. Consequently, an effective performance improvement can be expected from data compression schemes which can read and decompress the data at significantly higher speed than reading the uncompressed data. We make use of a brick-based compression layer fulfilling this requirement (Treib et al., 2012), yet we adapt it to support locally adaptive error control.

Since in particle tracing the interpolation errors accumulate and are transmitted to the calculated trajectories, we analyze—compared to the established interpolation scheme on the uncompressed data—the inaccuracies in the computed trajectories, which are caused by lossy compression.

Intuitively one might argue that lossy compression should not be considered, because it introduces an additional, non-acceptable error into particle tracing. On the other hand, in our application study the vector fields were simulated using a spectral method, meaning that the data values are a discrete sampling of a band-limited smooth function. Therefore, a ground truth interpolation exists—namely trigonometric interpolation—yet it is never used due to its high numerical complexity. Nevertheless it is clear that the established interpolation scheme already introduces an error, even though this error is generally accepted. As our major contribution we show that the additional inaccuracies caused by lossy data compres-

sion are in the same regions of variation in which the trajectories in the uncompressed field differ from the assumed ground truth trajectories. For the particular application this means that the trajectories extracted from the compressed data are as reliable as the trajectories usually used for analyzing the turbulence fields.

We focus on the analysis of the (spatial) interpolation error, because it is well known that interpolation is the major source of errors in numerical particle tracing in fully resolved turbulent flow fields. This is due to the fact that turbulent velocity fields are highly nonlinear. Since the time-step in turbulence simulations is commonly restricted to small values to enforce the Courant number stability condition, the time-stepping error in numerical integration is generally much less significant.

We use two vector-valued data sets describing turbulent flow fields to verify our approach. These data sets are the result of terascale turbulence simulations and originate from the JHU turbulence database cluster, which is publicly accessible at `http://turbulence.pha.jhu.edu`. Each is comprised of one thousand time steps of size 1024^3, making every time step as large as 12 GB (3 floating-point values per velocity sample). The data sets contain direct numerical simulations of magneto-hydrodynamic (MHD) turbulence and forced isotropic turbulence, and are called "MHD" and "Iso" in the following. For a description of the simulation methods used to compute these data sets let us refer to (Li et al., 2008).

2 RELATED WORK

We do not attempt here to survey the vast body of literature related to flow visualization approaches based on stream and path line integration because they are standard in flow visualization. For a thorough overview, however, let us refer to the reports by (Post et al., 2003), (Laramee et al., 2004), and (McLoughlin et al., 2010).

(Teitzel et al., 1997) put special emphasis on the investigation of the numerical integration error and the error introduced by interpolation. They conclude that an RK3(2) integration scheme provides sufficient accuracy compared to linear interpolation, but they do not consider higher-order interpolation methods. There is also a number of works dealing especially with accuracy issues of particle tracing in turbulence fields (Yeung and Pope, 1988; Balachandar and Maxey, 1989; Rovelstad et al., 1994). One of the conclusions was that Lagrange interpolation of order 4 to 6 provides sufficient accuracy, and it is therefore the established scheme in practice (cf. (Li et al., 2008)).

An important topic related to our method is data compression using transform coding. For a general overview of data compression techniques we refer to the book by (Sayood, 2005). The recent survey by (Balsa Rodriguez et al., 2013) provides a more focused treatment of techniques used in the context of volume visualization. Our GPU compression scheme builds upon previous work for performing wavelet-based vector field compression including Huffman and run-length decoding entirely on the GPU (Treib et al., 2012).

In previous work it has also been proposed to precompute and store particle trajectories for a number of prescribed seed points, and to restrict the visualization to subsets of these trajectories (Lane, 1994; Bruckschen et al., 2001; Ellsworth et al., 2004). In this way, all computation is shifted to the preprocessing stage, and storage as well as bandwidth limitations at runtime can be overcome. Conceptually, this approach can be seen as a kind of lossy data compression, where the seeding positions are quantized rather than the flow data itself.

Another possibility to overcome memory bandwidth limitations in particle tracing is to employ parallel computing architectures such as tightly coupled CPU clusters or supercomputers, providing larger memory capacities and I/O bandwidth. The two principal parallelization strategies for particle tracing are *parallelize-over-seeds* (PoS) and *parallelize-over-blocks* (PoB) (cf. (Pugmire et al., 2009)). In both strategies, the data set is partitioned into blocks, yet in PoS each processor dynamically loads those blocks required to trace the particles assigned to it, while in PoB the blocks are distributed across the processors and each only handles particles within its assigned blocks. Especially PoS can effectively take advantage of data compression for per-block node-to-node communication. A number of approaches have been presented to further improve PoS and PoB (Camp et al., 2011; Nouanesengsy et al., 2011; Peterka et al., 2011; Yu et al., 2007).

As reported e.g. in (Camp et al., 2011), particle tracing on compute clusters typically spends only a small fraction of the total time on the computation of particle traces. In many approaches, most of the time is spent on either node-to-node communication or memory I/O. It can be concluded that despite the used architecture there is a dire need for data compressions to increase the performance of particle tracing.

3 TURBULENT VECTOR FIELD COMPRESSION

Without data compression, the performance of particle tracing in large turbulence fields is vastly limited by disk I/O throughput. For instance, the computation of stream lines as shown in Fig. 1 in one single *uncompressed* time step involves a working set of almost 5 GB. Stream line integration takes roughly 45 seconds on our target architecture, of which over 98% are spent waiting for data from disk. Tracing the particles once the data is available takes only 1.2 seconds. By using our proposed compression scheme, stream lines can be extracted in less than 5 seconds, including disk I/O.

To avoid any additional error when using data compression, lossless compression schemes can be used in principle. However, for floating-point data—which is the internal format in which velocities are stored—the compression rate is usually quite modest. The lossless schemes proposed in (Isenburg et al., 2005; Lindstrom and Isenburg, 2006) compress the turbulence data to roughly $\frac{2}{3}$ of its original size. A decoding throughput of about 10 million floating point values per second is achieved, corresponding to over 600 ms for the decompression of a single 128^3 grid of 3D velocities. More sophisticated prediction schemes can slightly improve the compression rate (Fout and Ma, 2012), yet they come at lower throughput.

On the contrary, in (Treib et al., 2012) a lossy GPU compression scheme for vector data was shown to operate significantly above disk speed. The scheme is based on the discrete wavelet transform, followed by a quantization of wavelet coefficients and a final entropy coding of quantized coefficients. A decoding throughput of over 650 million floating-point values per second was achieved, at 3 bits per velocity vector and a signal to noise ratio above 45 db.

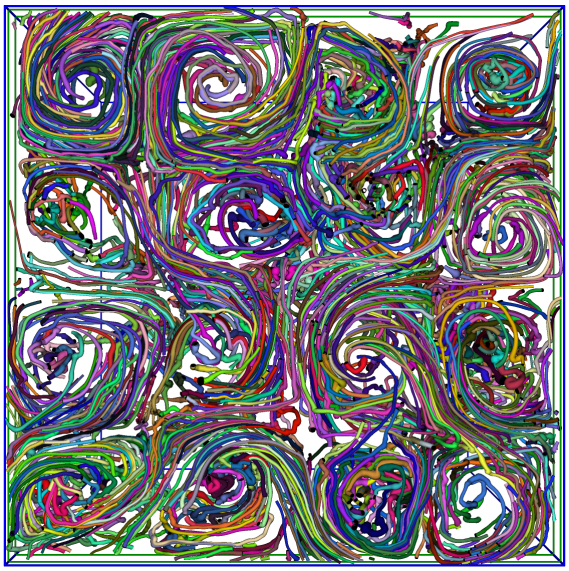

Figure 1: Stream lines in (*uncompressed*) MHD (1024^3).

3.1 Interpolation Error Estimate

When a lossy scheme for vector field compression is used, the reconstructed field is afflicted with some error compared to the initial field. At first, this seems to preclude lossy compression schemes in particle tracing, because the local reconstruction errors accumulate along the particle trajectories. On the other hand, this error has to be seen in relation to the error that is inherent to particle trajectories even when computed in the original data.

Even without compression the reconstructed samples are not exact in general, due to the interpolation which is used to reconstruct the data values from the initially given discrete set of samples. This interpolation makes assumptions on the continuous field which, in general, do not hold. As a consequence, it has to be accepted that the trajectories we compute numerically using interpolation diverge from those we would see in reality, even without compression.

It therefore makes sense to choose a compression quality so that the additional error introduced by the compression scheme is in the order of the error introduced by interpolation. It is worth noting, however, that without additional information about a data set it is impossible to accurately compute or even estimate the interpolation error. In some cases, theoretical error bounds depending on higher-order derivatives of the continuous function can be given; see, for instance, (Fout and Ma, 2013) for such a bound when linear interpolation is used. On the other hand, the derivatives of the continuous function are typically not known exactly. In that case, such bounds themselves come with some uncertainty. In addition, even with exact knowledge of the derivatives, they often overestimate the actual error significantly (Zheng et al., 2010).

Therefore, we have adopted a different approach to estimate the interpolation error: We take the difference between interpolation results from a reference high-order interpolator and a lower-order interpolator as an estimate for the error in the low-order interpolator. For the two discrete turbulence data sets we analyze in this work (Iso and MHD), an exact interpolator is known. Due to the pseudo-spectral method that was used to simulate the turbulent motion (Li et al., 2008), the velocity field is guaranteed to be band-limited in the Fourier sense. As a consequence,

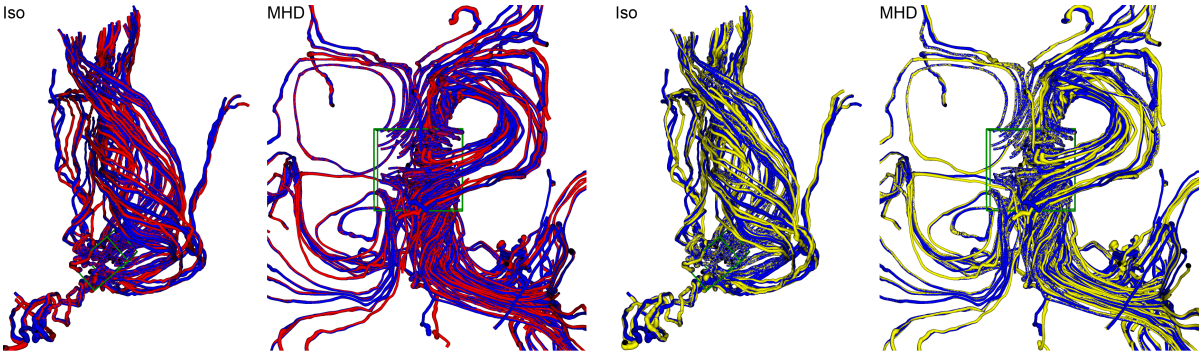

Figure 2: Path lines in the Iso and stream lines in the MHD data sets. Left: Particle trajectories using (ground truth) trigonometric (blue) and (established) Lagrange6 (red) interpolation in the original data for velocity sampling. Right: Particle trajectories using trigonometric interpolation (blue) and Lagrange6 interpolation (yellow) in the compressed data. The yellow traces appear to be of similar accuracy as the red lines.

Fourier or trigonometric interpolation using trigonometric polynomials of infinite support gives exact velocity values between grid points (Rovelstad et al., 1994). Due to efficiency reasons, however, what is used in practice for particle tracing is an interpolation scheme of "sufficiently high order" which resembles Fourier interpolation, e.g. Lagrange6. For instance, in Fig. 2 (left) the trajectories using trigonometric and Lagrange6 interpolation are compared. It is worth noting that even though in the turbulence community it is usually agreed that Lagrange6 is of sufficient accuracy for particle tracing, significant deviations from the ground truth can be observed.

The interpolation error over the whole volume for a given interpolator can now be computed: We upsample the volume to 4 times the original resolution using the interpolator under consideration as well as the reference interpolator. The RMS of the difference between the upsampled volumes then is a good approximation of the average error introduced by the interpolation. Since trigonometric interpolation has to be evaluated globally, to generate the interpolant in a computationally efficient way, we have adopted the following approach: First, we perform a fast Fourier transform (FFT) on the flow field using the FFTW library (Frigo and Johnson, 2005). In the frequency domain, we then quadrupel the data resolution in each dimension by zero padding. Finally, an inverse FFT is performed to generate a flow field of 4 times the original resolution. This field agrees with the original field at every 4th vertex, and the other vertices lie on the trigonometric interpolant between the original data samples. Fig. 3 illustrates FFT-based upsampling in 1D, when the data resolution is doubled. Generating the 4096^3 trigonometric interpolant from a 1024^3 velocity field in this way takes about 1.5 hours including disk I/O. Given the 4096^3 trigonometric interpolant, evaluating the interpolation errors in a 1024^3 velocity field for the listed interpolation schemes takes another 2 hours including disk I/O.

3.2 Error-Guided Data Compression

Equipped with the average interpolation error for a given interpolation scheme, we can locally adapt the compression error so that it is equal to or falls below the interpolation error. This is achieved by choosing a quantization step in the data compression scheme so that the prescribed error bound is not exceeded. In the wavelet-based compression scheme we use, the average error is roughly equal in magnitude to the quantization step and, thus, the acceptable error is a reasonable choice for the quantization step. Table 1 lists the RMS interpolation errors in both data sets for a number of different interpolation schemes. To verify that the lossy compression does not unduly affect the interpolant, we have computed the interpolation errors a second time after compression, comparing the reconstructed volumes to the original reference solution. It can be seen that by setting the quantization step equal to the RMS interpolation error, the error is increased by less than 50% in all cases. It is worth mentioning that performing the same test with an upsampling factor of only 2 instead of 4 yields almost identical results (within 5% of the listed numbers). It indicates that the discrete computation approximates the actual interpolation error very closely. This is expected, as the reference interpolant is by definition band-limited with respect to the original resolution, so no high-frequency deflections can occur between the original grid points.

It remains to show that the accumulation of the additional quantization errors does not introduce significantly larger regions of variation in the trajectories. A first experiment can be seen in Fig. 2 (right), where the trajectories computed on the compressed

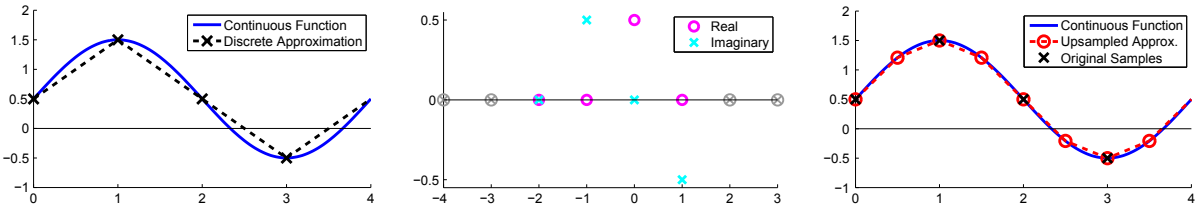

Figure 3: FFT-based upsampling process. Left: A periodic band-limited function with a period of 4, and its discrete approximation sampled at a frequency of 1. Middle: FFT coefficients of the function in magenta and cyan. Because the input function was real, the coefficients have a Hermitian symmetry. The coefficients are padded with zeros to the left and right, corresponding to higher frequencies with an amplitude of zero. Right: The inverse FFT of the padded coefficients results in a higher-resolution approximation to the continuous function. Note that the even grid points of the upsampled approximation agree with the original grid points.

field using Lagrange6 interpolation are compared to the ground truth trajectories. Compared to Fig. 2 (left), the deviations seem to be in the same order of variation. A detailed quantitative accuracy analysis is given in the following Section.

Table 1: Root-mean-square interpolation errors for different interpolation schemes in two turbulent flow fields before (*orig*) and after compression (*comp*), compared to the reference trigonometric interpolant. The interpolation error has been evaluated in a grid of four times the original resolution.

	Iso (range: 6.67)		MHD (range: 2.77)	
Interpolation	orig	comp	orig	comp
Lagrange8	0.86E-3	1.26E-3	3.48E-4	4.97E-4
Lagrange6	1.10E-3	1.60E-3	4.52E-4	6.32E-4
Lagrange4	1.71E-3	2.41E-3	7.20E-4	9.63E-4
Linear	5.15E-3	6.65E-3	2.29E-3	2.81E-3

4 EVALUATION

To evaluate the accuracy of the resulting trajectories, we have conducted a number of experiments where the proposed lossy compression scheme was used. In the following, we first introduce the error metrics we use to analyze the accuracy of the computed trajectories.

4.1 Error Metrics

Due to errors induced by the employed interpolation scheme and by lossy compression, a trajectory may gradually diverge from the ground truth over time. To evaluate the accuracy of computed trajectories, an error metric is required to quantitatively measure the difference between two trajectories starting at the same seed point.

One obvious metric is the maximum or average distance between trajectories $s_0(u)$, $s_1(u)$ along their parameter u. In addition, several metrics exist which measure some kind of distance between two curves, such as the (discrete) Fréchet distance (Eiter and Mannila, 1994) and the distance under dynamic time warping (DTW). While the Fréchet distance corresponds to a type of maximum distance, the DTW distance is akin to an average distance. Both disregard the u parametrization and instead are concerned only with the shape of the curves. All these metrics measure the distance along the complete trajectories. However, once two particles have diverged by some critical distance, their further behavior depends only on the characteristics of the flow field: They might diverge further or even converge again, but this provides no insight into the accuracy of the trajectory computation. Therefore, we introduce a new metric taking this into account, which we call the (clamped) divergence rate. Instead of measuring a distance between trajectories, it computes the rate at which they diverge. Given two trajectories $s_0(u), s_1(u)$ over a parameter interval $[u_0, u_{\max}]$, we define their divergence rate as

$$d_{s_0,s_1} := \frac{\mathrm{dist}(u_{\mathrm{div}})}{u_{\mathrm{div}} - u_0}, \text{ where}$$

$$\mathrm{dist}(u) := \|s_0(u) - s_1(u)\| \text{ and}$$

$$u_{\mathrm{div}} := \max\left\{u \in [u_0, u_{\max}] \;\middle|\; \forall \tilde{u} \in [u_0, u] : \mathrm{dist}(\tilde{u}) \leq \Delta s\right\}.$$

u_{div} is the last point along the trajectories at which they have not yet diverged by more than Δs. In the following experiments, we have set the critical distance Δs equal to the grid spacing.

Our definition of the trajectory divergence rate is similar in spirit to the idea of the finite-size Lyapunov exponent (FSLE) (Aurell et al., 1997). The FSLE measures the time it takes for two particles, initially separated only by an infinitesimal ε, to diverge by some given distance, usually specified as a multiple of ε. A fundamental difference is that in our case both trajectories start at exactly the same position, and

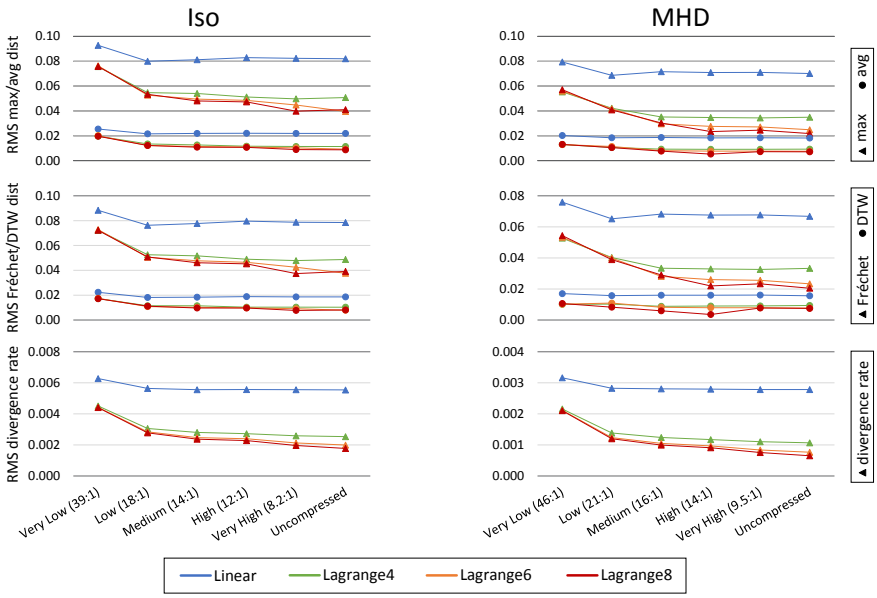

Figure 4: Accuracy of stream lines vs. compression quality using different interpolation schemes. Accuracy is reported as the root mean square (RMS) of the individual trajectory distances (see Section 4.1), computed against trajectories traced using the (approximate) trigonometric reference interpolation.

we measure their divergence as an absolute distance rather than relative to their initial separation.

4.2 Error Analysis

To compare the accuracy of particle trajectories computed in the original and compressed data sets, and via different interpolation schemes, a reference solution is required to which the trajectories can be compared. For the used turbulence data sets, trigonometric interpolation is known to be exact. Since evaluating the trigonometric interpolant during particle tracing is impracticable, we have upsampled the data sets to four times the original resolution (see Section 3.1) as the ground truth. Particle trajectories traced in the upsampled versions using 16th order Lagrange interpolation then act as the reference solution. While this is not equivalent to true trigonometric interpolation in the original data, the remaining error is expected to be negligible since the difference between the two times and four times upsampled versions is already very small (cf. Section 3.1).

For the accuracy analysis of computed trajectories, we have generated a set of 4096 seed points in each data set, that were randomly distributed over the entire domain. Particles were traced from the seed points through different versions of the data sets: The upsampled reference version, the original uncompressed data, and compressed versions at different compression rates. The quantization steps for the compressed versions were chosen equal to the errors in linear and Lagrange4/6/8 interpolation as listed in Table 1. In addition, we generated one high-quality compressed version of each data set, where the quantization step was set to half the Lagrange8 interpolation error. The compressed file sizes and compression rates (as the ratio of original size to compressed size) are listed in Table 2.

To minimize the impact of inaccuracies due to numerical integration errors, in all of our experiments we used the Runge-Kutta method by (Dormand and Prince, 1980). The method provides a 5th order solution and a 4th order error estimate, which is used for adaptive step size control. The error tolerance for step size control was reduced until the accuracy of the results did not improve any further.

Table 2: File sizes and compression factors. For *Very Low*, *Low*, *Medium*, and *High*, the quantization step was chosen equal to the error in linear and Lagrange4/6/8 interpolation, resp. (cf. Table 1); for *Very High*, to half the error in Lagrange8.

	Iso		MHD	
Quality	size	factor	size	factor
Uncompressed	14.7 GB	–	14.7 GB	–
Very High	1.79 GB	8.21	1.55 GB	9.48
High	1.25 GB	11.8	1.06 GB	13.9
Medium	1.08 GB	13.6	942 MB	16.0
Low	843 MB	17.9	712 MB	21.1
Very Low	387 MB	38.9	331 MB	45.5

Fig. 4 provides the main results of our accuracy analysis. The graphs show the root mean square (RMS) of the average, maximum, Frchet, and DTW distance, as well as the divergence rate, over all trajectories for different compression rates and interpolation schemes. For reference, the grid spacing is approximately 0.00614 in both data sets. The most prominent finding is that linear interpolation performs very poorly and eclipses the errors introduced at even the highest compression rates. The differences between the other interpolation schemes are comparatively small; as expected, with some advantage of the higher-order schemes. All distance metrics give qualitatively similar results. However, all metrics except for our novel divergence rate display a significant amount of noise in the results. This is caused by a few individual trajectories with a very large distance to their reference. These trajectories have a very large impact on the RMS distance, but actually carry little information on the accuracy of the results, as explained in Section 4.1. The divergence rate, on the other hand, handles such trajectories well.

The most important observation with regard to the lossy compression is that when the quantization step is chosen smaller than the interpolation error (e.g. Lagrange4 interpolation and a compression quality of "Medium" or higher), the additional error introduced by the compression is extremely small. For example, switching from Lagrange6 to Lagrange4 interpolation has a larger impact on the accuracy than switching from uncompressed data to the "High" compression quality in both data sets.

5 PARTICLE TRACING SYSTEM

We have integrated the presented compression layer into an out-of-core GPU particle tracing system to facilitates an interactive visual exploration of large scale turbulence fields on commodity PC hardware. In the following we will first give a brief system overview. Next, we will evaluate its performance and, finally, we will compare its performance to previous systems that have employed large compute clusters to extract integral lines from large scale (unsteady) flow fields.

5.1 System

Our proposed system for out-of-core particle tracing takes as input a sequence of 3D velocity fields, each field representing the state of a flow field at a different point in time. We assume that the values in each field are given on a Cartesian grid. In a preprocess, each grid is partitioned into a set of equally-sized bricks. A halo region is added around each brick to allow proper interpolation at brick boundaries. The bricks are compressed before being stored sequentially on disk.

At runtime, the computation of particle trajectories is performed on the GPU. For that, bricks which are required to perform the numerical integration are requested from the CPU and cached in their compressed form in main memory. The compression reduces disk bandwidth requirements and allows us to cache a large number of bricks. For use on the GPU, the compressed brick data is uploaded into GPU memory and immediately decompressed. In the current implementation we use bricks of size 128^3 each (including a halo region of size 4). We have found that this size provides the best trade-off between locality of access and storage overhead for the halo regions. Multiple bricks are stored in a *brick atlas* whose size is chosen based on the amount of available GPU memory.

Fluid particles are advected in parallel on the GPU to exploit memory bandwidth and computational capacities. We use the CUDA programming API and issue one thread per particle, grouped into thread blocks of size 128. Each thread advects the position of its particle while the required flow data is available in the *brick atlas*. Since the set of bricks which are required to perform the computation of all trajectories does not fit into GPU memory in general, only a subset can be made available at a time. An additional *index* buffer stores the mapping from a brick index (a tuple consisting of a spatial index and an id for the time step) to position in the brick atlas. If a particle enters a region for which the respective brick is missing in the *brick atlas* (flagged by a -1 in the index buffer), the GPU requests this brick for the next round of tracing. This is realized by atomically incrementing the corresponding entry in an additionally *requests* buffer which maps spatial regions and point in time to brick indices.

The GPU stops when all particles a) must stop because they are waiting for a brick to be uploaded to the GPU, b) have been advected for a predefined amount of time, or c) have been advanced by a fixed maximum number of steps (64 in the current implementation). The CPU then downloads the *requests* buffer and determines the bricks to be uploaded next into the atlas–prioritized by their respective request count. When path lines are traced, requested bricks corresponding to an earlier time step are prioritized higher so that all path lines advance in time at roughly the same speed, thus reducing multiple uploads of the same data during the execution of one multi-pass advection step. If the systems main memory is too small to hold all necessary bricks, the CPU fetches the brick

data from disk and replaces an existing brick in memory based on an LRU (Least Recently Used) caching strategy. The CPU also tracks the earliest time step which was requested globally and pages out all brick time steps older than that, since they will not be visited again by the current particles. The requested bricks are then sent to the GPU and particle tracing is restarted. The process is finished once all particles have either reached their maximum age or left the domain.

Special care has to be taken whenever a particle moves close to a brick boundary. In this case it has to be ensured that all velocity values required in the integration step are available in the current brick. While we employ a halo region to ensure that all values required by the support of the interpolation kernel are available, for a multi-stage integration method not only the initial particle location, but also all intermediate stages of the integrator must reside within the admissible area. This can be guaranteed by adapting the maximum integration step size Δt appropriately.

5.2 Performance Evaluation

The performance of any particle tracing system depends on a multitude of factors, such as the characteristics of the data set, the number and placement of seeding locations, and the total integration time. This makes an exhaustive performance evaluation and comparison to other approaches fairly difficult. Instead, we tried to capture the typical performance characteristics of our system. For both data sets, we investigated the following two scenarios:

1. **Sparse:** This is the same scenario that was used for pursuing the accuracy analysis. 4096 seeding locations are distributed uniformly in the domain, and particles are traced for 2.5 and 5 time units in Iso and MHD, respectively.
2. **Dense:** This scenario models an interactive exploration. 1024 seed points are placed within a small box with an edge length of 10% of the domain size. The particles are traced over 5 and 10 time units in Iso and MHD, respectively.

All timings were measured on a PC with an Intel Core i5-3570 CPU (quad-core, 3.4 GHz) with 8 GB of DDR3-1600 RAM, equipped with an NVIDIA GeForce GTX 680 GPU with 4 GB of video memory. The size of the brick atlas was set to 2 GB of video memory.

We have traced particles starting from the selected seed points in both the uncompressed and the compressed data sets to demonstrate the performance gains that can be achieved via compression. In all experiments, Lagrange6 interpolation was performed; the particle integration times are about $3\times$ higher with Lagrange8, and about $4\times$ lower with Lagrange4 interpolation. When particle tracing was performed on the compressed data, the timings refer to the "High" compression quality. The decompression times for other compression rates differ only very slightly. To measure the impact of disk I/O, we have run every benchmark a second time, so that all required data was already cached in CPU memory. With uncompressed data, however, this was only possible for the dense scenario; in case of sparse particle seeding, the size of the working set exceeded the available CPU memory (even for stream lines extracted from a single time step). Table 3 lists the time required for running each scenario, and Table 4 lists the sizes of the corresponding working sets.

It can be seen that the use of compression allows us to trace thousands of characteristic lines within seconds in the dense seeding scenario. In the sparse seeding case, the required time is around an order of magnitude higher. The reason becomes clear when looking at the size of the working sets, which are larger by roughly the same factor in those cases.

Without compression, the overall system performance is clearly limited by disk bandwidth. In particular, in the sparse scenario, the working set was so much larger than main memory (cf. Table 4) that some bricks had to be loaded from disk multiple times. Even when all required data is already cached in CPU memory (which was only possible in the dense scenario), the performance of the compressed and uncompressed cases is very similar—the runtime overhead caused by the additional decompression step is very minor.

Table 3: Times in seconds for computing stream lines, both for the cached case (*C*) and the un-cached case including disk access times (*U*). Individual times for uploading the data to the GPU (*Upl*, including decompression), particle integration (*Int*), and disk I/O (*IO*, overlapping *Upl* and *Int*) are listed separately.

Scenario	Quality	U	C	Upl	Int	IO
Iso dense	High	2.3	1.4	0.8	0.6	1.9
	Uncomp	18.6	1.3	0.7	0.6	17.8
Iso sparse	High	21.6	16.4	12.4	3.8	14.9
	Uncomp	156.9	n/a	10.7	3.8	156.4
MHD dense	High	3.4	2.3	1.4	0.8	2.8
	Uncomp	26.3	2.3	1.4	0.8	25.6
MHD sparse	High	19.9	16.2	11.8	4.2	13.6
	Uncomp	139.7	n/a	10.4	4.2	138.8

Table 4: Working set sizes in both compressed (*High*) and uncompressed (*Uncomp*) form. Also shown is the number of bricks in the working set (*#B*) as well as the number of brick uploads during particle integration (*#U*).

Scenario	High	Uncomp	#B	#U
Iso dense	165.9 MB	2155.5 MB	92	92
Iso sparse	1280.6 MB	15058.9 MB	728	1341
MHD dense	243.8 MB	3231.0 MB	138	155
MHD sparse	1095.5 MB	15066.5 MB	729	1298

It is clear that when tracing path lines, the working sets are much larger because often many different time steps of the same spatial brick are required. In particular, the temporal distance between successive time steps is extremely small in both data sets (0.002 time units for Iso, 0.0025 for MHD). Because of this, the time required for path line computation is spent almost exclusively on disk-to-CPU data transfer and GPU decompression, and only a negligible amount of time is spent on the actual particle integration (less than 1% in our tests). For example, tracing a set of path lines with the dense seeding configuration through Iso takes about 6 minutes, with a working set size of over 25 GB of compressed data. In the uncompressed setting, the working set comprises over 300 GB. Correspondingly, tracing these path lines in the uncompressed data set takes almost an hour, and most of that time is spent on disk I/O. In MHD, the time required for path line tracing is similar; in all cases, the time scales proportionally to the working set size.

5.3 Comparison to Previous Work

To the best of our knowledge, all previous techniques for particle tracing in very large flow fields have employed large compute clusters. Pugmire et al. (Pugmire et al., 2009) have used 512 CPUs to trace 10K stream lines in two steady flow fields comprising 512 million grid cells each. They report wall times of 10 to 100 seconds. Camp et al. (Camp et al., 2011) later improved those timings to a few seconds for tracing thousands of stream lines on 128 cores. Nouanesengsy et al. (Nouanesengsy et al., 2011) achieve timings between 10 and 100 seconds using 4096 cores for the computation of 256K stream lines in regular grids of up to 1.67 billion grid points, but at the cost of an expensive preprocess. Peterka et al. (Peterka et al., 2011) report computation times of about 20 seconds using 8192 cores for 128K stream lines in a 1024^3 steady flow, and several minutes for 32K lines in a $2304 \times 4096 \times 4096$ steady flow. In contrast to all other mentioned approaches, they have also addressed large *unsteady* flow fields. In a $1408 \times 1080 \times 1100 \times 32$ unsteady flow, the processing time is several minutes for 16K path lines on 4096 cores.

While an exact performance comparison to our technique is not possible due to the different data sets and interpolation/integration schemes used, an order-of-magnitude comparison reveals that our method achieves competitive timings to the previous approaches in many cases, particularly in dense seeding scenarios, while making use of only a single desktop PC.

All in all it can be said that due to the use of an effective compression scheme, the performance of particle tracing in extremely large flow fields can be improved significantly. It is clear that due to the immense working set that is required when computing path lines, fully interactive rates cannot be achieved in this case.

6 CONCLUSION

In this paper we have discussed the application of lossy compression of large scale turbulent vector fields in the context of particle tracing to overcome bandwidth limitations and storage requirements. And we have demonstrated how this compression layer can be integrate into an out-of-core GPU particle tracer, to facilitate an interactive visual exploration of tera scale data sets on a desktop PC. This gives rise to integrating the exploration process into a researchers daily workflow as a means to validate hypotheses. In a number of experiments we have demonstrated that compared to interpolation errors in the reconstruction of the velocity field, the compression errors do not significantly affect the accuracy of the computed trajectories. In the statistical sense, the quality of the computed trajectories remains in the same order. A performance analysis indicates that such a system achieves a throughput that is comparable to that of previous systems running on high-performance architectures.

The most challenging future avenue of research will be the investigation of the effect of lossy data compression in scenarios other than turbulence research. The question will be whether lossy data compression can also be applied to other flow fields without unduly affecting the accuracy of the resulting trajectories. The main difficulty is that for most flow fields a “correct” interpolation scheme is not available, so the interpolation error can not be estimated accurately. However, different criteria might be found to steer the compression quality, e.g. given confidence intervals for the velocity values.

ACKNOWLEDGEMENTS

The work was partly funded by the European Union under the ERC Advanced Grant 291372: Safer-Vis - Uncertainty Visualization for Reliable Data Discovery. The authors want to thank Charles Meneveau from Johns Hopkins University and Tobias Pfaffelmoser from TUM for helpful discussions and constructive criticism.

REFERENCES

Aurell, E., Boffetta, G., Crisanti, A., Paladin, G., and Vulpiani, A. (1997). Predictability in the large: an extension of the concept of Lyapunov exponent. *J. Physics A: Mathematical and General*, 30(1):1–26.

Balachandar, S. and Maxey, M. R. (1989). Methods for evaluating fluid velocities in spectral simulations of turbulence. *J. Computational Physics*, 83(1):96–125.

Balsa Rodriguez, M., Gobbetti, E., Iglesias Guitián, J., Makhinya, M., Marton, F., Pajarola, R., and Suter, S. (2013). A survey of compressed GPU-based direct volume rendering. In *Eurographics 2013 - STARs*, pages 117–136.

Bruckschen, R., Kuester, F., Hamann, B., and Joy, K. I. (2001). Real-time out-of-core visualization of particle traces. In *Proc. IEEE 2001 Symp. on Parallel and Large-Data Visualization and Graphics*, pages 45–50.

Camp, D., Garth, C., Childs, H., Pugmire, D., and Joy, K. (2011). Streamline integration using MPI-hybrid parallelism on a large multicore architecture. *IEEE Trans. Vis. Comput. Graphics*, 17(11):1702–1713.

Dormand, J. R. and Prince, P. J. (1980). A family of embedded Runge-Kutta formulae. *J. Computational and Applied Mathematics*, 6(1):19–26.

Eiter, T. and Mannila, H. (1994). Computing discrete Frèchet distance. Technical Report CD-TR 94/64, Technische Universität Wien.

Ellsworth, D., Green, B., and Moran, P. (2004). Interactive terascale particle visualization. In *IEEE Visualization*, pages 353–360.

Fout, N. and Ma, K.-L. (2012). An adaptive prediction-based approach to lossless compression of floating-point volume data. *IEEE Trans. Vis. Comput. Graphics*, 18(12):2295–2304.

Fout, N. and Ma, K.-L. (2013). Fuzzy volume rendering. *IEEE Trans. Vis. Comput. Graphics*, 18(12):2335–2344.

Frigo, M. and Johnson, S. G. (2005). The design and implementation of FFTW3. *Proc. IEEE*, 93(2):216–231.

Isenburg, M., Lindstrom, P., and Snoeyink, J. (2005). Lossless compression of predicted floating-point geometry. *Computer Aided Design*, 37(8):869–877.

Lane, D. A. (1994). UFAT—a particle tracer for time-dependent flow fields. In *IEEE Visualization*, pages 257–264.

Laramee, R. S., Hauser, H., Doleisch, H., Vrolijk, B., Post, F. H., and Weiskopf, D. (2004). The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum*, 23(2):203–221.

Li, Y., Perlman, E., Wan, M., Yang, Y., Meneveau, C., Burns, R., Chen, S., Szalay, A., and Eyink, G. (2008). A public turbulence database cluster and applications to study Lagrangian evolution of velocity increments in turbulence. *J. Turbulence*, 9:N31.

Lindstrom, P. and Isenburg, M. (2006). Fast and efficient compression of floating-point data. *IEEE Trans. Vis. Comput. Graphics*, 12(5):1245–1250.

McLoughlin, T., Laramee, R. S., Peikert, R., Post, F. H., and Chen, M. (2010). Over two decades of integration-based, geometric flow visualization. *Computer Graphics Forum*, 29(6):1807–1829.

Nouanesengsy, B., Lee, T.-Y., and Shen, H.-W. (2011). Load-balanced parallel streamline generation on large scale vector fields. *IEEE Trans. Vis. Comput. Graphics*, 17(12):1785–1794.

Peterka, T., Ross, R., Nouanesengsy, B., Lee, T.-Y., Shen, H.-W., Kendall, W., and Huang, J. (2011). A study of parallel particle tracing for steady-state and time-varying flow fields. In *Parallel Distributed Processing Symposium (IPDPS)*, pages 580–591.

Post, F. H., Vrolijk, B., Hauser, H., Laramee, R. S., and Doleisch, H. (2003). The state of the art in flow visualisation: Feature extraction and tracking. *Computer Graphics Forum*, 22(4):775–792.

Pugmire, D., Childs, H., Garth, C., Ahern, S., and Weber, G. H. (2009). Scalable computation of streamlines on very large datasets. In *Proc. Conf. on High Performance Computing, Networking, Storage and Analysis*, pages 16:1–16:12.

Rovelstad, A. L., Handler, R. A., and Bernard, P. S. (1994). The effect of interpolation errors on the Lagrangian analysis of simulated turbulent channel flow. *J. Computational Physics*, 110(1):190–195.

Sayood, K. (2005). *Introduction to Data Compression*. Morgan Kaufmann Publ. Inc., 3rd edition.

Teitzel, C., Grosso, R., and Ertl, T. (1997). Efficient and reliable integration methods for particle tracing in unsteady flows on discrete meshes. In *Visualization in Scientific Computing*, pages 31–41.

Treib, M., Bürger, K., Reichl, F., Meneveau, C., Szalay, A., and Westermann, R. (2012). Turbulence visualization at the terascale on desktop PCs. *IEEE Trans. Vis. Comput. Graphics*, 18(12):2169–2177.

Yeung, P. K. and Pope, S. B. (1988). An algorithm for tracking fluid particles in numerical simulations of homogeneous turbulence. *J. Computational Physics*, 79(2):373–416.

Yu, H., Wang, C., and Ma, K.-L. (2007). Parallel hierarchical visualization of large time-varying 3D vector fields. In *Proc. ACM/IEEE Conf. on Supercomputing*, pages 24:1–24:12.

Zheng, Z., Xu, W., and Mueller, K. (2010). VDVR: Verifiable volume visualization of projection-based data. *IEEE Trans. Vis. Comput. Graphics*, 16(6):1515–1524.