Turbulence Visualization at the Terascale on Desktop PCs

Marc Treib, Kai Bürger, Florian Reichl, Charles Meneveau, Alex Szalay, and Rüdiger Westermann

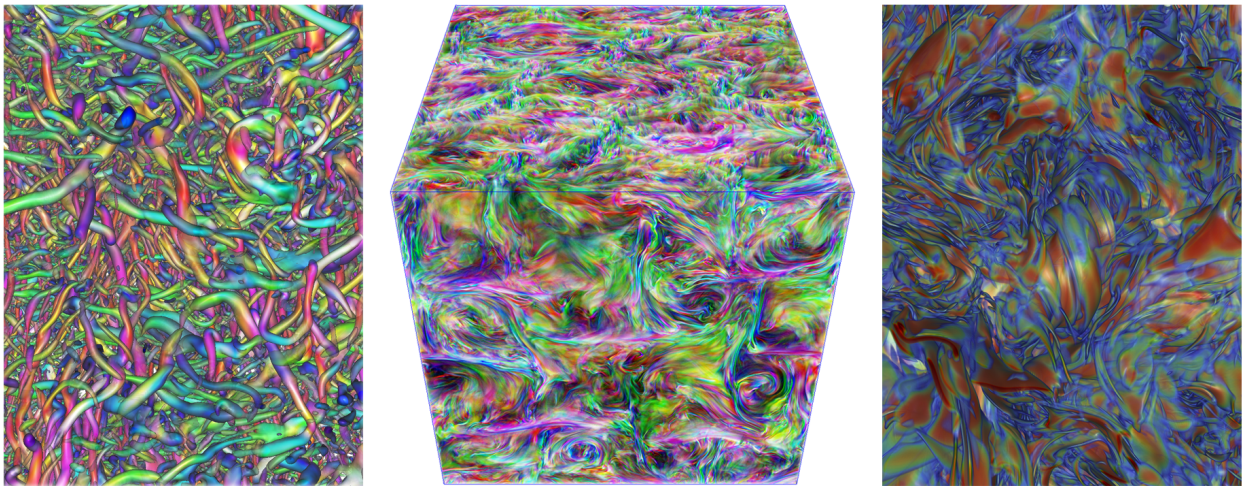

Fig. 1. Visualizations of structures in 1024^3 turbulence data sets on 1024×1024 viewports, directly from the turbulent motion field. Left: Close-up of iso-surfaces of the Δ_{Chong} invariant with direct volume rendering of vorticity direction inside the vortex tubes. Middle: Direct volume rendering of color-coded vorticity direction. Right: Close-up of direct volume rendering of R_S. The visualizations are generated by our system in less than 5 seconds on a desktop PC equipped with 12 GB of main memory and an NVIDIA GeForce GTX 580 graphics card with 1.5 GB of video memory.

Abstract—Despite the ongoing efforts in turbulence research, the universal properties of the turbulence small-scale structure and the relationships between small- and large-scale turbulent motions are not yet fully understood. The visually guided exploration of turbulence features, including the interactive selection and simultaneous visualization of multiple features, can further progress our understanding of turbulence. Accomplishing this task for flow fields in which the full turbulence spectrum is well resolved is challenging on desktop computers. This is due to the extreme resolution of such fields, requiring memory and bandwidth capacities going beyond what is currently available. To overcome these limitations, we present a GPU system for feature-based turbulence visualization that works on a compressed flow field representation. We use a wavelet-based compression scheme including run-length and entropy encoding, which can be decoded on the GPU and embedded into brick-based volume ray-casting. This enables a drastic reduction of the data to be streamed from disk to GPU memory. Our system derives turbulence properties directly from the velocity gradient tensor, and it either renders these properties in turn or generates and renders scalar feature volumes. The quality and efficiency of the system is demonstrated in the visualization of two unsteady turbulence simulations, each comprising a spatio-temporal resolution of 1024^4. On a desktop computer, the system can visualize each time step in 5 seconds, and it achieves about three times this rate for the visualization of a scalar feature volume.

Index Terms—Visualization system and toolkit design, vector fields, volume rendering, data streaming, data compression.

1 Introduction

Hydrodynamic turbulence is one of the most thoroughly explored phenomena among complex multi-scale physical systems. It has important applications in engineering thermo-fluid systems, in the geosciences and environmental transport, even in astrophysics. In recent years, high performance computing [20] and new experimental measurement techniques [22, 42] applied to the study of various types of turbulent flows have enabled significant progress. Yet, modeling and understanding turbulent flows remains a scientifically deep, technologically relevant, but fundamentally unsolved, problem.

One grand challenge that significantly increases the complexity of turbulence analysis is turbulence's inherently vectorial and tensorial structure: one describes turbulent flows using velocity and vorticity vector fields, and velocity gradient and stress tensor fields. Some of the most salient features of turbulent flows have emerged from an examination of the velocity gradient tensor. It is defined according to

$$A_{ij} = \frac{\partial u_i}{\partial x_j},$$

where we use index notation; $u_i(\mathbf{x},t)$, $i = 1,2,3$ denote the three components of the velocity vector field (which in turbulent flows depend on position vector $\mathbf{x}$ and time t). Such gradient fields of fluid velocity provide a rich characterization of the local quantitative and qualitative behavior of flows, which is evident from the linear approximation in the neighborhood of an arbitrary point. Since $\mathbf{A}$ is a second-rank tensor, it has nine components (in three dimensions) and these contain rich information about the local properties of the flow. Since the tensor $\mathbf{A}$ encodes much information through each of its matrix elements, analysis of its properties is quite challenging. Therefore, certain scalar

- *Marc Treib, Kai Bürger, Florian Reichl, and Rüdiger Westermann are with the Technische Universität München, Munich, Germany. E-mail: {treib,buerger,westermann}@tum.de, reichlf@in.tum.de.*
- *Charles Meneveau and Alex Szalay are with the Johns Hopkins University, Baltimore, Maryland, USA. E-mail: {meneveau,szalay}@jhu.edu.*

Manuscript received 31 March 2012; accepted 1 August 2012; posted online 14 October 2012; mailed on 5 October 2012.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

quantities that characterize basic properties of **A** have been proposed and are often analyzed as scalar fields, e.g. the vorticity magnitude, the dissipation rate, the angle between vorticity and the strain-rate eigenvectors, or the magnitude of the rotation tensor, to name just a few.

One of the primary challenges in turbulence research is to endow the traditional statistical analysis of metrics (e.g. the average of an alignment angle) with more geometrical insights into the overall structure of turbulence affecting more than one specific property. Even though a number of different feature metrics are known, no single feature can alone explain all relevant effects. This means that different features must be explored simultaneously and in an interactive fashion, to be seen in relation to each other. Only then can one proceed with evaluating more meaningful statistical metrics. For example, one would like to visualize the high vorticity, high rotation, or high Q regions in the flow, but in relation with the alignments of the local strain-rate eigen-directions, or together with another scalar field such as R. Particularly the question whether the geometric trends in the small-scale turbulence structures are also shared by the coarse-grained (or filtered) velocity gradient tensor plays a determining role in turbulence research. A visual indication of the relationship between velocity increments and the filtered velocity gradients at coarser scales can enable further insights into the complicated multi-scale behavior of turbulence.

The visual exploration of many different intrinsic features of turbulence, however, is very challenging. The major reason is that the fine-scale structures are fully resolved only at the very highest resolution in both space and time. For instance, in the current paper we will address the visualization of two terascale turbulence simulations, each 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). These data sets contain direct numerical simulations of forced isotropic turbulence (see Fig. 1, left) and magneto-hydrodynamic turbulence (see Fig. 1, middle and right), respectively. For a detailed descriptions of the simulation and database methods used let us refer to [26] and the web page at *http://turbulence.pha.jhu.edu*. For such data it is simply not feasible to precompute multiple feature volumes and inspect these volumes simultaneously, in particular because the number of potentially interesting features and scales is so large. Furthermore, to be able to faithfully represent even the smallest features in the data, highly accurate reconstruction schemes are necessary which work directly on the turbulence field by reconstructing features in turn during visualization.

As a consequence, visualization systems necessary to explore the full turbulence spectrum require an innovative approach that provides extreme I/O capabilities, combined with computational resources that allow for an efficient feature reconstruction and rendering. Since the data to be visualized is so large that even storing one single time step in CPU memory can become problematic, bandwidth limitations in paging the data from disk become a major bottleneck.

Following the requirements in turbulence visualization, we have developed a new holistic approach which combines scalable data streaming and feature-based visualization with novel hardware and software solutions, such as a deep integration of GPU computing. We employ the capabilities of wavelet-based data compression and on-the-fly GPU data decoding and encoding to reduce memory access and bandwidth limitations. Because our approach reduces disk access and CPU-GPU data transfer, it is suitable for the analysis of small-scale turbulence structures on desktop systems which are not equipped with large main memory. To preserve even the finest structures, feature extraction is embedded into the visualization process, based on the direct computation of vector field derivatives and on-the-fly evaluation of gradient tensor-based feature metrics.

Our system distinguishes from previous approaches for visualizing turbulent flow fields in that it eases bandwidth and memory limitations throughout the entire visualization pipeline. In particular, the system

- compresses vector data at very high fidelity,
- works on the compressed data up to the GPU, using on-the-fly GPU decompression and rendering,
- enables caching of derived feature volumes via on-the-fly GPU compression,
- provides multi-scale feature visualization via on-the-fly gradient tensor evaluation.

Our paper is structured in the following way: First, we review previous systems and algorithms for the visualization of large volumetric data sets. We then give an overview of our system, including its internal structuring as well as the basic functionality in a nutshell. Here we aim at giving information about what the system provides and how this is achieved, but we do not answer the question why the particular choices have been made. This question is addressed in the upcoming section, where we motivate our design decisions and discuss tradeoffs involved in making our system practical for visualizing large turbulence simulations. This also involves the demonstration of some advanced features which are made possible by these choices. Finally, we describe the streaming and visualization performance of our system and discuss its preprocessing costs.

2 Related Work

Previous efforts in large volume visualization can be classified into two major categories: a) Parallelization and b) data compression and out-of-core strategies. There is a vast body of literature on parallelization strategies for volume visualization on parallel computer architectures and a comprehensive review is beyond the scope of this work; however, some of the most recent works have addressed volume rendering on both GPU [8, 30] and CPU [18] clusters.

A different avenue of research has addressed the visualization of large volumetric data on desktop PCs. These works employ out-of-core techniques to dynamically load only the required part of a data set into memory, and many employ some form of compression to reduce the immense data volume. Vitter [41] provides a general overview including detailed analyses of many out-of-core algorithms. The most recent works focussing on the direct rendering of large-scale volume data [6, 14] employ an octree of volume bricks. During rendering, the octree is traversed on the GPU and visited nodes are tagged for refinement or coarsening. The tags are read back to the CPU which then updates the GPU working set accordingly. Such approaches allow for on-demand streaming of data and efficient rendering in a single pass, provided that all data required for the current view is available in GPU memory. In turbulent flow fields, however, using a lower-resolution approximation of the velocity data results in a significant distortion of the extracted features and is thus not admissible. Specifically for large flow data, Ellsworth et al. [7] present a particle-based visualization system that precomputes a large number of particle traces which can then be displayed interactively.

In the following, we review the most popular compression options in the context of volume rendering. For a general overview of data compression, we refer to the book by Sayood [34].

Lossless: Lossless compression typically employs some form of prediction to exploit spatial and temporal redundancy in the data. The prediction remainders are then compressed via any general-purpose compression approach. To the best of our knowledge, all existing work employing lossless compression in the context of volume rendering has addressed only integer-valued data [11, 12]. Outside of volume rendering, some fast lossless floating-point compressors exist [2, 27]. However, in particular for floating-point data, lossless compression usually achieves only quite modest compression rates.

Hardware-supported formats: Some special fixed-rate compression formats such as S3TC are implemented directly by graphics hardware. This means that rendering, including hardware-supported interpolation, is possible directly from the compressed form, thus reducing GPU memory requirements. By adding a second, CPU-based compression stage, the compression rate can be improved [31, 32]. However, the fixed-rate first stage allows little or no control over the quality vs. compression rate trade-off, and these formats lack support for floating-point data. Additionally, while decompression is extremely fast, the compression step is usually quite involved.

Vector quantization: In vector quantization, a data set is represented by a small codebook of representative values and, for each data point,

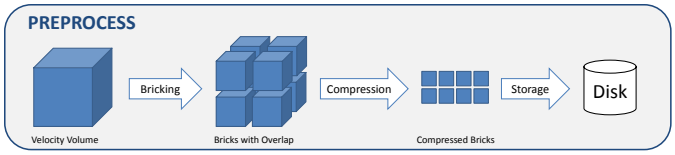

Fig. 2. Preprocessing pipeline.

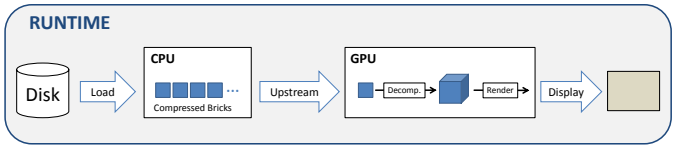

Fig. 3. Basic system data flow at run-time.

an index into this codebook. This allows for very fast decoding on the GPU via a single indirection [10, 35]. By employing deferred filtering [9], hardware-supported texture filtering becomes possible. However, the construction of a good codebook is extremely time-consuming, and it is difficult to satisfy the very high quality requirements of our application.

Transform coding: Transform coding approaches are used in most popular image and video compression schemes, such as the various JPEG and MPEG standards. The basic idea is to express the input data as coefficients to a set of basis functions with the goal of contracting most of the energy into few coefficients, so that most coefficients have very small values. In a following quantization step, these small coefficients are reduced to zero and need not be stored. The remaining coefficients are typically further compressed using an entropy coding scheme such as Huffman or arithmetic coding. The most commonly used transforms are the discrete cosine transform (DCT) and the discrete wavelet transform (DWT), both of which have been applied to volume rendering [28, 45], [15, 16, 33, 43]. Woodring et al. [44] analyze the application of JPEG 2000 compression to a large climate simulation. For data with little or no local correlation, Lakshminarasimhan et al. [25] present an alternative approach based on coefficient reordering and spline fitting.

3 System Functionality, Algorithms, and Features

Our approach begins with a sequence of 3D turbulent motion fields, each given on a Cartesian grid. In a preprocess, each vector field is partitioned into a set of equally sized bricks. An overlap between adjacent bricks enables proper interpolation at brick boundaries. Every brick is compressed separately and written to disk. The preprocess is outlined in Fig. 2.

3.1 Compression Algorithm

Once the bricked volume representation has been constructed, a wavelet-based scheme is used for compression, including run-length and Huffman encoding of the coefficient stream. We use the GPU compression scheme recently proposed by Treib et al. [39], which has been tailored explicitly to support both encoding and decoding on current GPU architectures. The wavelet compression builds upon the fast CUDA implementation of the discrete wavelet transform (DWT) by van der Laan et al. [40].

The compression algorithm is slightly modified to handle vector data, yet the basic stages remain mostly unchanged. In the first stage, a hierarchical DWT is performed separately on each component of the velocity vectors using the CDF 9/7 wavelet [5]. The floating-point wavelet coefficients C_i are quantized into integer values c_i via standard scalar dead-zone quantization, i.e., $c_i = \text{sign}(C_i)\,\lfloor |C_i|/\Delta_l \rfloor$, where Δ_l is the quantization step that is used at level l of the wavelet pyramid. Because the coefficients at coarser scales carry more energy than the coefficients at smaller scales, the quantization steps are decreased with increasing scale, i.e., starting at the finest level $l = 0$ with a user-defined step size Δ_0, on subsequent levels the step size is set to $\Delta_l = \Delta_0 \,/\, 2^l$. Here, Δ_0 provides control over the compression rate and reconstruction quality. The quantized wavelet coefficients are finally concatenated into a sequential coefficient stream in scan-line order. A run-length encoder followed by a Huffman encoder convert the coefficient stream into a highly compact form.

3.2 Visualization Algorithm

For visualizing the 3D vector field, we use GPU ray-casting on the bricked volume representation [24]. At every sample point along a ray, one or multiple scalar features are derived from the velocity gradient tensor (see Section 3.3), and the respective values are mapped to color and opacity. The velocity gradient tensor is computed on-the-fly via central differences between interpolated velocity values. The system supports tri-linear interpolation for fast previewing purposes and tri-cubic interpolation [36] for high-quality visualization. The visual difference between tri-linear and tri-cubic interpolation is demonstrated in Fig. 4. For shading purposes, gradients are approximated locally by central differences on six additional feature samples.

The bricks are traversed on the CPU in front-to-back order. If the compressed representation of the current brick is not residing in CPU memory, it is loaded from disk and cached in RAM using a LRU strategy. After all data for the current time step has been loaded, the CPU starts prefetching subsequent time steps asynchronously. The brick is then streamed to the GPU, where a CUDA kernel is executed to decompress the data and store it in texture memory. Our system leverages GPU texture memory to take advantage of hardware-supported texture filtering. Streaming to the GPU works asynchronously, meaning that the transmission stalls neither the CPU nor the GPU.

Once the data has been decompressed, a CUDA ray-casting kernel is launched. It invokes one thread for every pixel covered by the screen-aligned rectangle enclosing the projected vertices of the brick's bounding box. Each thread first determines if the respective view ray intersects the bounding box and terminates if no hit was detected. Otherwise, the current frame buffer content at the pixel position is read, and the brick data is re-sampled along the ray to obtain the color and opacity contribution to be accumulated with the current values. Early ray termination is performed whenever the opacity has reached a value of 0.99. Fig. 3 illustrates the basic system data flow at run-time.

Based on a set of basic rendering modalities, i.e., direct volume rendering (DVR) including iso-surface rendering and scale-invariant volume rendering [23], our system supports a number of different visualization options, for example, the simultaneous rendering of iso-surfaces of multiple turbulence features, or a combination of different techniques such as DVR and iso-surface rendering. Furthermore, a comparative visualization of the same feature at different scales is supported by enabling simultaneous operations on the initial and filtered data (see Section 4.3). By incorporating shader dependencies, the visualization of features can be made dependent on the existence or properties of other features. Some examples of different visualization options are shown in Fig. 1 and Fig. 5.

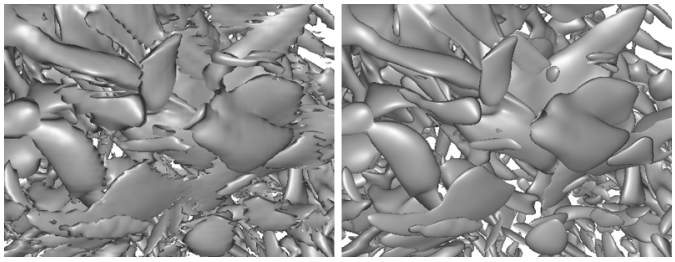

Fig. 4. Comparison of tri-linear (left) vs. tri-cubic (right) filtering when rendering iso-surfaces. With tri-linear interpolation, the silhouettes of high-frequent iso-surfaces are poorly resolved.

3.3 Turbulence Features

In our system, turbulence features are derived from the velocity gradient tensor. The gradient fields of fluid velocity provide a rich characterization of the local quantitative and qualitative behavior of flows, which is evident from the linear approximation in the neighborhood of

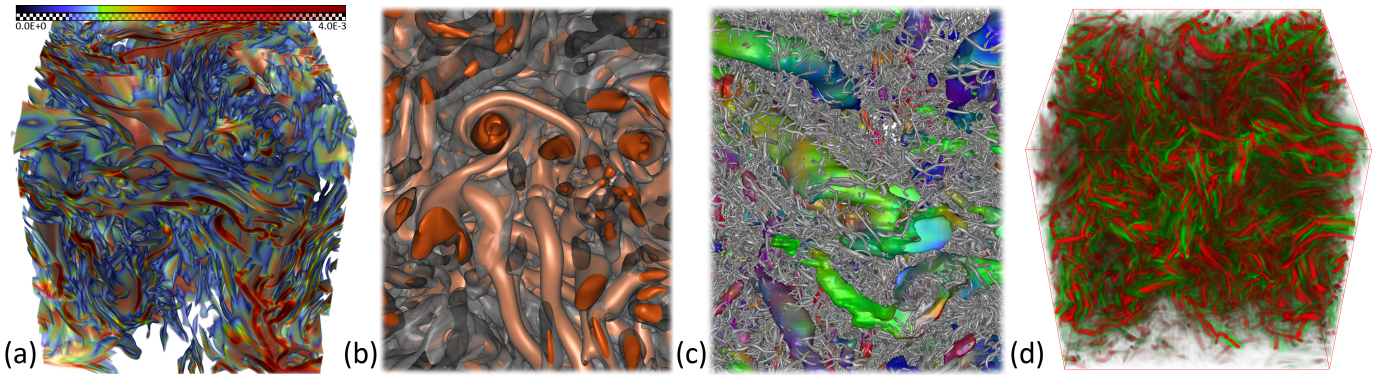

Fig. 5. Turbulence visualizations. (a) Direct volume rendering of E. (b) Two semi-transparent iso-surfaces of Q_{Hunt}. (c) Fine-scale iso-surfaces (gray) and coarse-scale iso-surfaces colored by vorticity direction. (d) Direct volume rendering of λ_2; negative values are red, positive values green.

an arbitrary point,

$$\mathbf{x}_0 : u_i(\mathbf{x},t) = u_i(\mathbf{x}_0,t) + A_{ij}(\mathbf{x}_0,t)(x_j - x_{0j}) + \ldots .$$

Since $\mathbf{A}$ is a second-rank tensor, it has nine components (in three dimensions) and these contain rich information about the local properties of the flow. The decomposition

$$A_{ij} = S_{ij} + \Omega_{ij}, \text{ where } S_{ij} = \frac{1}{2}\left(A_{ij} + A_{ji}\right), \; \Omega_{ij} = \frac{1}{2}\left(A_{ij} - A_{ji}\right),$$

is commonplace and separates $\mathbf{A}$ into its symmetric part (the strain-rate tensor $\mathbf{S}$) and its antisymmetric part (the rotation-rate tensor $\mathbf{\Omega}$). The tensor $\mathbf{S}$ has three real eigenvalues λ's that in incompressible flow add up to zero, and if they are different (non-degenerate case) the tensor $\mathbf{S}$ has three orthogonal eigenvectors that define the principal axes of $\mathbf{S}$. These indicate directions of maximum rate of fluid extension ($\lambda_\alpha > 0$) and contraction ($\lambda_\gamma < 0$), and an intermediate fluid deformation that can be either extending or contracting in the third direction. $\mathbf{\Omega}$ describes the magnitude and direction of the rate of rotation of fluid elements and is simply related to the vorticity vector $\boldsymbol{\omega} = \nabla \times \mathbf{u}$. Since the tensor $\mathbf{A}$ encodes much information through each of its matrix elements, an analysis of its properties is quite challenging. Therefore, certain scalar quantities that characterize basic properties of $\mathbf{A}$ have been proposed and are often analyzed as scalar fields.

For example, it has been found convenient to define the following five scalar invariants [3, 29]:

$$Q = -\frac{1}{2}\text{Trace}(\mathbf{A}^2) = -\frac{1}{2}A_{ij}\,A_{ji} := -\frac{1}{2}\sum_{i=1}^{3}\sum_{j=1}^{3} A_{ij}\,A_{ji},$$

$$R = -\frac{1}{3}A_{ij}\,A_{jk}\,A_{ki},$$

$$Q_S = -\frac{1}{2}S_{ij}S_{ji}, \quad R_S = -\frac{1}{3}S_{ij}S_{jk}S_{ki}, \quad V^2 = S_{ij}S_{ik}\omega_j\omega_k.$$

Additional commonplace, Galilean invariant vortex definitions involve non-trivial combinations of $\mathbf{A}$, $\mathbf{S}$ and $\mathbf{\Omega}$, such as the Q_{Hunt} and Δ_{Chong}-criterion [4, 17, 19]:

$$Q_{Hunt} = \frac{1}{2}\left(|\mathbf{\Omega}|^2 - |\mathbf{S}|^2\right) > 0, \quad \Delta_{Chong} = \left(\frac{Q_{Hunt}}{3}\right)^3 + \left(\frac{\det\mathbf{A}}{2}\right)^2 > 0.$$

Further vortex classifications employ additional information from the vorticity, or eigenvalues through an eigendecomposition of symmetric tensors. For example, the λ_2-criterion [21] identifies vortex regions by $\lambda_2 < 0$, where λ_2 is the second largest eigenvalue of the symmetric tensor $\mathbf{S}^2 + \mathbf{\Omega}^2$. Another option is the enstrophy production, which is defined as $E = S_{ij}\omega_i\omega_j$.

One striking observation in turbulence research was the preferential vorticity alignment found by Ashurst et al. [1]. They observed that the most likely alignment of the vorticity vector $\boldsymbol{\omega}$ was with the intermediate eigenvector $\boldsymbol{\beta}_S$, the direction corresponding to the eigenvalue λ_β that could be either positive or negative. For a random structureless gradient field, no such preferred alignment would be expected, and on naïve grounds one might have expected the vorticity to align with the most extensive straining direction instead. Therefore, the observations generated sustained interest in the problem of alignment properties of the vorticity field and relationships with features related to the strain-rate tensor—e.g. its eigenvectors' directions. For this reason, our system provides mappings of the vector components of $\boldsymbol{\omega}$ (or one of the eigenvectors $\alpha_\mathbf{S}, \beta_\mathbf{S}, \gamma_\mathbf{S}$ of the strain-rate tensor) to RGB colors during volume ray-casting.

Besides analyzing the small-scale turbulence structures, a substantial amount of research has been devoted to the statistical features of velocity increments in the inertial range [13, 37]. In particular, it has been shown that a relationship between velocity increments at scale Δ and coarse-grained (or filtered) velocity gradients $\tilde{\mathbf{A}} = G_\Delta * \mathbf{A}$ at scale Δ can be established. Here, G_Δ is a convolution kernel—usually an averaging box filter—of characteristic scale ℓ. To enable a visual multi-scale analysis of turbulence, our system allows simultaneously extracting and visualizing features from filtered velocity fields at different (user-selected) scales. As filtering and differentiation are linear operations, filtering is performed on the velocity vector field instead of the gradient tensor field.

4 Design Decisions and Tradeoffs

We now consider some of the decisions made in the implementation of our system that make it suitable for visualizing large turbulence data. In particular, we want to emphasize the possible tradeoffs that allow the user to choose between highest quality and highest speed. As we will show, despite the careful design of the system with regard to the application-specific requirements, not always can it respond interactively to the user inputs. This is because of the extreme amounts of data to be processed and the complex shaders to be evaluated. However, our results demonstrate a system performance that facilitates an interactive exploration for most of the supported visualization options.

The bricked data representation the system builds upon is necessary to keep the chunks of data that are processed at run-time manageable. In addition, the bricked representation has the advantage of enabling view frustum culling, resulting in a considerable reduction of the data to be streamed to the GPU. The integration of occlusion culling is also possible, but the turbulence structures are typically so small and scattered that no significant gain can be expected. We also want to mention that level-of-detail rendering strategies as they are typically employed in volume ray-casting have not been considered, because the continuous transition between multiple scales of turbulence in one image has been determined inappropriate by turbulence researchers.

To avoid access to neighboring bricks in tri-linear/tri-cubic data interpolation and gradient computation, a 4-voxel-wide overlap is stored around each brick. Thus, the smaller the bricks, the more additional memory is required to store the overlaps. On the other hand, the larger the bricks, the fewer disk seek operations have to be performed for reading the bricks from disk. Consequently, we make the bricks as

large as possible, yet we consider that a certain number of decompressed bricks (at least 4) should fit into GPU memory as a working set. We chose a brick size of 248^3 in our system, so that a brick including the overlap can be stored in a texture of size 256^3. This results in a memory overhead of about 10%.

4.1 Feature Reconstruction

The visualization system by default reconstructs turbulence features directly from the velocity field during ray-casting. It would also be possible to precompute certain (scalar) feature volumes in a preprocess and to visualize these volumes. However, such an approach is problematic in the current scenario. First, it would cause a significant increase of the overall memory consumption. Second, the system would become inflexible to the extent of the precomputed features, prohibiting an interactive steering of the feature extraction processes. Third, quality losses are introduced by re-sampling a scalar feature volume instead of a direct feature reconstruction.

Two examples demonstrating the quality differences are shown in Fig. 6. The images clearly reveal that certain fine-scale structures can no longer be reconstructed from the scalar feature volumes. Even though the principal shapes are still maintained, a detailed analysis of the bending, stretching, merging, and separating behavior of the turbulence features is no longer possible.

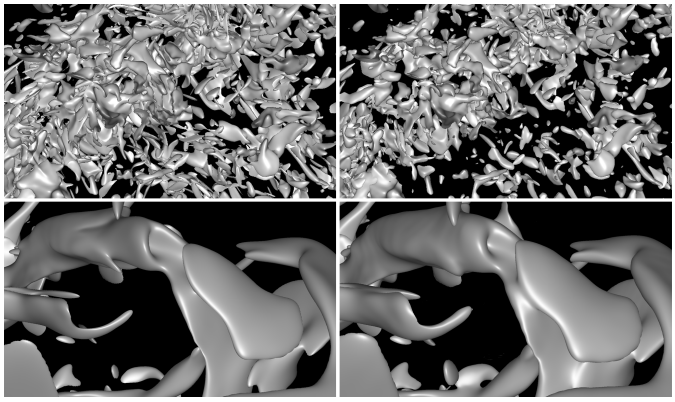

Fig. 6. Features reconstructed from the turbulent motion field (left) and from a precomputed scalar feature volume (right).

On the other hand, ray-casting a feature volume can be a viable approach to obtain an overview of the turbulence structures. Therefore, our system supports the construction and storage of scalar feature volumes for fast previewing purposes (see also Section 5). Even though the construction of such a volume on the GPU is straightforward using the system's functionality, this volume might be too large to be stored on the GPU in uncompressed form. The requirement to tackle this problem has significantly steered our selection of the compression scheme to be used in the system.

4.2 Lossy Compression

Because of the extreme data volumes to be handled by our system, the reduction of this volume becomes one of the most important requirements. Without any reduction, expensive disk-to-CPU data transfer becomes the major performance bottleneck since only a very small portion of an entire turbulence sequence can be stored in main memory. To meet this requirement, we have embedded a data compression layer into the system. This layer encapsulates a compression scheme that can be used for arbitrarily-sized bricks and, thus, can be integrated seamlessly into the system architecture.

In the decision which compression to use, the following aspects have been considered. First of all, it is required that no features will be destroyed due to the compression, and that possible quality losses do not affect the features' shapes significantly. For floating-point data, compression schemes like S3TC and vector quantization [34] do not adhere to this constraint. Second, the compression rate must be so high that the data can be streamed from disk and decompressed at a rate that keeps pace with the data processing speed. Especially due to this requirement, lossless compression schemes are problematic. In general, lossless schemes, for instance as proposed in [2, 27], can only achieve a rather moderate compression rate.

A last consideration arises from the particular requirement of our system to generate scalar feature volumes on-the-fly for the purpose of fast previewing. This functionality goes hand in hand with the possibility to efficiently compress the generated feature volumes on the GPU, so that they can be efficiently streamed to the CPU and buffered in RAM. While the compressed volumes could also be stored on the GPU, the time required to transfer the compressed data between the CPU and the GPU is negligible compared to the compression time. Thus, all generated data is always buffered on the CPU. To support this option, an alternative data flow as illustrated in Fig. 7 is realized in our system. This rules out compression schemes such as vector quantization, because the construction of the vector codebook in the coding phase can not be performed at sufficient rates in general.

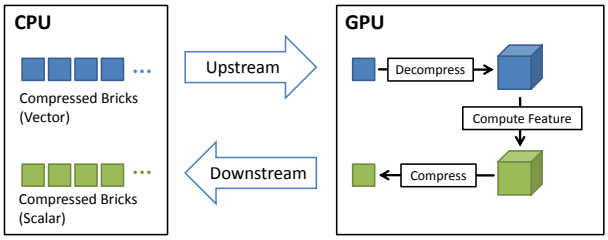

Fig. 7. Alternative data flows at run-time support construction and reuse of derived feature volumes.

Lossy compression schemes based on the discrete wavelet transform, in combination with coefficient quantization and entropy coding, are well known to achieve very high compression rates at high fidelity [38]. Compression schemes based on transform coding also have a long tradition in visualization, for instance to reduce memory and bandwidth limitations in volume visualization [16, 33, 43, 45]. However, only with the possibility to perform the entire compression pipeline on the GPU [39]—including encoding *and* decoding—can the full potential of wavelet-based compression be employed for large data visualization.

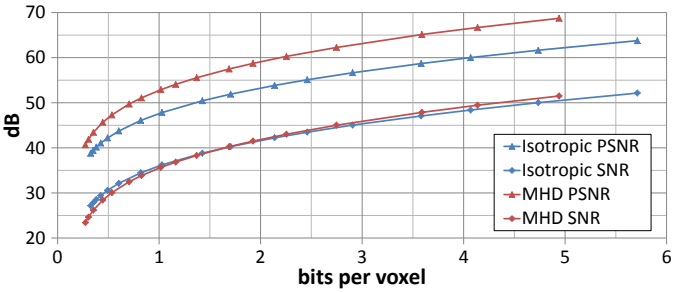

Fig. 8. Rate-distortion curves showing dB (P)SNR vs. bits per voxel for two different turbulence fields.

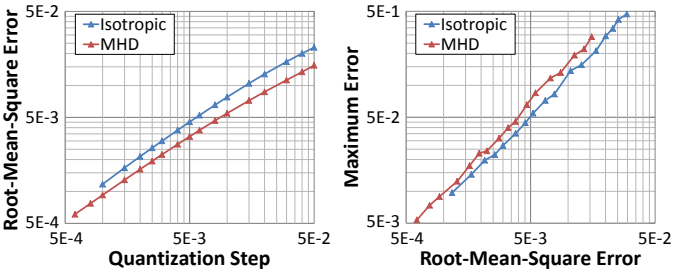

Fig. 9. Graphs showing RMSE vs. quantization step, and maximum error vs. RMSE for two different turbulence fields. RMSE and quantization step are of the same order of magnitude, while the maximum error is consistently about 1 order of magnitude larger than RMSE.

To assess the compression rate and reconstruction quality of the wavelet-based GPU coder, we have performed tests using two different turbulence simulations, each consisting of time steps of size 1024^3. On each brick a three-level DWT was performed, and the wavelet coefficients were compressed as described. Rate-distortion curves in (P)SNR vs. bits per voxel (where each voxel contains a 3-component floating-point vector) for both data sets are given in Fig. 8. In addition, Fig. 9 plots RMS error vs. quantization step as well as maximum error vs. RMS error. The graphs demonstrate that the user can directly control the compression error by choosing an appropriate quantization step size. The rate-distortion curves demonstrate the high reconstruction quality of the wavelet-based compression. On the other hand, they do not provide an intuitive notion of the visual quality. Therefore, in Figs. 10 and 11, we compare the visual quality of the rendered structures in the compressed motion fields to those in the original fields. For comparison purposes we use different compression rates.

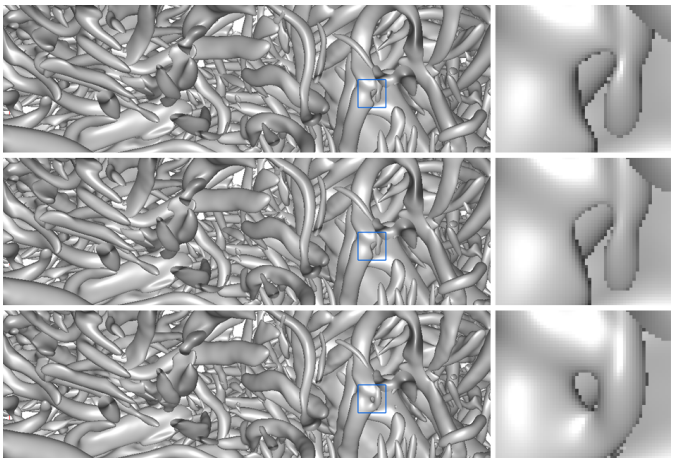

Fig. 10. Visual quality comparison for an iso-surface in Q_{Hunt} in the isotropic turbulence data set. Structures are reconstructed from the original vector field (top), and a compressed version at 3.0 bpv (middle) and 1.3 bpv (bottom).

Even though it is clear that the compression quality depends on the visualization parameters, such as the transfer function, the selected iso-value, and on the feature that is visualized, a component-wise wavelet transform can very effectively reduce the memory consumption, yet it achieves a very high reconstruction quality. In particular, in the middle images in Figs. 10 and 11 the structures are reproduced at almost no visual difference at a remarkable compression rate of 32:1.

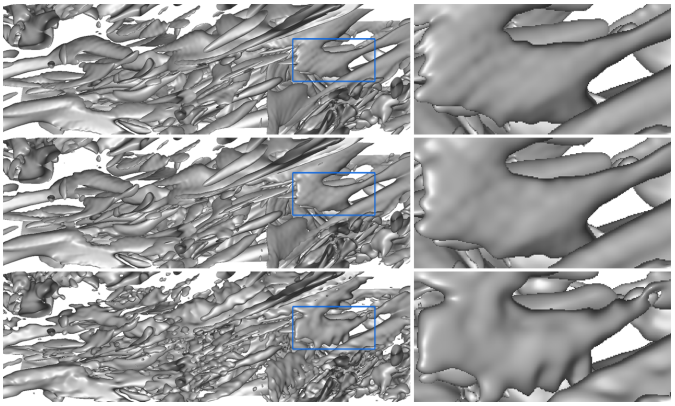

Fig. 11. Visual quality comparison for an iso-surface in R_S in the MHD turbulence data set. Structures are reconstructed from the original vector field (top), and a compressed version at 3.0 bpv (middle) and 1.3 bpv (bottom).

Let us finally mention that the selected compression scheme can in principle be extended to also exploit the motion coherence between successive time steps for increasing the compression rate. The basic idea is to not compress every time step separately, but rather to encode the differences between a time step and one or more preceding (or, in some cases, succeeding) time steps. Popular video compression algorithms such as MPEG typically employ block-based motion compensation, and this approach has been applied to volume compression as well [15]. However, any temporal prediction scheme obviously requires access to one or more other time steps to use as input. If these time steps can not be held in RAM in uncompressed form, then such a prediction necessarily triggers additional decompression steps and thus significantly increases the processing time. In our application, the possible gain in compression rate does not justify the increase in compression/decompression time. Therefore, our system compresses each time step separately.

4.3 Multi-Scale Analysis

As we have emphasized in the introduction, one challenging endeavor in turbulence research is the analysis of the shape and evolution of structures at different scales. To enable such an analysis, it is necessary to filter the velocity field using a low-pass filter. A linear convolution filter is used in practice. It can then be instructive to compare the original data with the filtered version, or multiple instances filtered with different radii. Both the initial and the filtered data are made accessible to the shader, and they are ray-cast simultaneously. Many options for combined visualizations are now possible, for example, iso-surfaces for different iso-values and with different colorings (Fig. 5 (c)), or the conditional visualization of fine features depending on coarse features (Fig. 12).

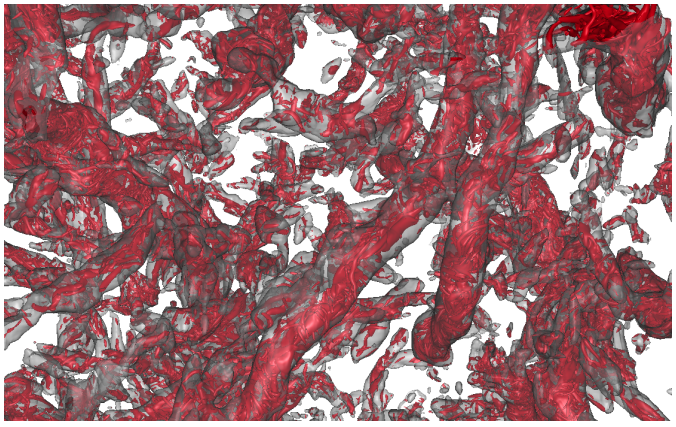

Fig. 12. Multi-scale turbulence analysis in a "focus+context" manner. Iso-surfaces in the fine-scale data (red) are extracted only within iso-surfaces of the coarse-scale version.

Because a large range of scales may contain relevant features, these scales can not be precomputed but must be determined interactively at run-time. Furthermore, the interesting scales are often quite large, requiring filter radii of 20 and more voxels, so that an on-the-fly filtering during rendering is not feasible. The large filter radii also make a separate filtering of each brick impossible, because values from adjacent bricks are required in the convolution. Unfortunately it is also impossible to keep all 26 neighbors of one brick in GPU memory at the same time, such that the bricks required for filtering have to be streamed and accessed sequentially. As a consequence, every brick needs to be loaded from the CPU and decompressed multiple times.

To avoid this, we have restricted our system to the execution of separable filters, i.e. filters that can be expressed as 3 successive 1D filters, one along each coordinate axis. In this case, only 3 filtering passes are required, and in each pass only the 2 neighbors along the current filter direction need to be available (as long as the filter radius is not larger than the brick size). In each pass, the bricks are traversed in an order which ensures that each brick needs to be decompressed only once. Fig. 13 illustrates this ordering.

All intermediate results and the final filtered result are compressed in turn on the GPU and buffered in CPU memory. Consequently, additional losses will occur, besides those which are introduced by the

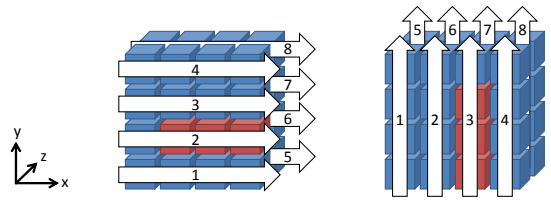

Fig. 13. Brick ordering during separable filtering in x and y dimension (left and right, respectively). The z dimension is analogous. One exemplary working set during each filtering pass is indicated in red.

encoding of the initial vector field. On the other hand, because the data becomes smoother and smoother after each filtering pass, at the same compression rate ever better reconstruction quality can be achieved. In all of our examples, the additional losses were only very minor, and noticeable differences between single-pass and multi-pass filtering on the compressed vector field could hardly be observed. This is demonstrated in Fig. 14 for a particular feature iso-surface in the isotropic turbulence data set.

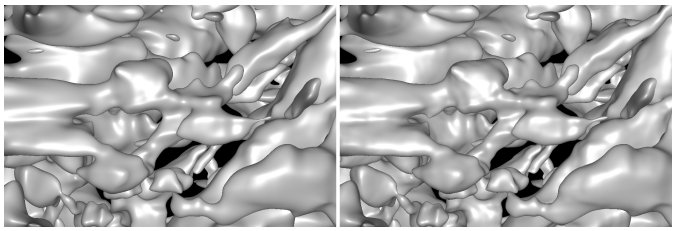

Fig. 14. λ_2 iso-surface in an isotropic turbulence simulation. Left: A 3D smoothing filter with support 25 was applied to the compressed vector field in one pass. Right: The same filter as before was used, but now the filter was separated and filtering was performed in 3 passes. After each pass, the intermediate results were compressed and buffered on the CPU.

5 Performance

In this section, we evaluate the performance of all components of our system and provide accumulated timings for the most time-consuming operations. All presented timings were performed on a desktop PC with an Intel Xeon E5520 CPU (quad core, 2266 MHz), 12 GB of DDR3-1066 RAM, and an NVIDIA GeForce GTX 580 graphics card with 1.5 GB of video memory. We used a standard hard disk providing a sustained data rate of about 100 MB/s.

Our statistics are based on the two terascale turbulence simulations referenced in the introduction and shown in Fig. 1. Each comprises one thousand turbulent motion fields of size 1024^3. The vector samples are stored as 3 floating-point values. Both data sets were compressed to 3.0 bpv at a compression rate of 32:1. The compression rates for individual bricks ranged from 53:1 to 22:1, depending on their content. The performance of all operations in our system scales roughly linearly in the spatial resolution of the data, so the system performance for data sets of different sizes can be easily extrapolated from the numbers listed below.

Although preprocessing time is not as important as visualization time, it is still significant for the practical visualization of very large data sets. On our test system, preprocessing takes about 3 minutes per time step. The majority of this time is spent reading the uncompressed data from disk; the compression of one time step—once stored in CPU memory—takes only about 10 seconds. This time includes the upload of the raw data to the GPU, the compression on the GPU, and the download of the compressed data to the CPU.

Table 1 summarizes typical timings for visualizing one (compressed) brick of size 256^3 and one (compressed) time step consisting of 5^3 bricks. We give separate times for data streaming and compression, performing data operations on the GPU, and rendering. For comparison, the statistics also include timings for an uncompressed data set. Rendering was always to a 1024×1024 viewport. The entire volume was shown so that view-frustum culling did not have any effect—in zoomed-in views where some bricks can be culled, decompression and ray-casting are faster accordingly. Furthermore, high transparency was assigned to the structures to eliminate any effects of early ray termination. Which feature metric was used had no significant effect on the performance. Since the visualization performance for different time steps and for the two different data sets was very similar, they are not listed separately in the table.

Table 1. Timings for individual system components. Where appropriate, values are given as min–avg–max.

Data streaming	per brick	per time step
Read from disk	35–60–88 ms	4.2 s
Upload & decompress (vector-valued)	18–32–39 ms	3.0 s
Compress & download (vector-valued)	48–105–230 ms	10.3 s
Upload & decompress (scalar)	7–14–16 ms	1.3 s
Compress & download (scalar)	23–43–56 ms	4.2 s
GPU processing	per brick	per time step
Compute metric	28 ms	2.1 s
Filter (per pass)	3.9 ms	0.3 s
Rendering	per brick	per time step
Ray-cast, on-the-fly feature, tri-linear	6–25–35 ms	2.1 s
Ray-cast, on-the-fly feature, tri-cubic	27–150–205 ms	12.1 s
Ray-cast, multi-scale, tri-cubic	54–305–415 ms	24.4 s
Ray-cast, precomp. feature, tri-linear	0.8–2.3–3.9 ms	0.2 s
Ray-cast, precomp. feature, tri-cubic	1.7–6.5–10 ms	0.6 s
Data streaming (uncompressed)	per brick	per time step
Read from disk	1.9 s	2.3 min
Upload or download (vector-valued)	34 ms	3.2 s
Upload or download (scalar)	11 ms	1.1 s

Table 2. Aggregate timings for some common scenarios.

Scenario	startup	per frame
Preview rendering (precomp. feature, tri-linear)	9.3 s	1.5 s
Standard rendering (on-the-fly feature, tri-linear)	0.0 s	4.9 s
HQ rendering (on-the-fly feature, tri-cubic)	0.0 s	15.1 s
HQ rendering w/ multi-scale analysis	40.8 s	29.8 s

One can see that the upload and decompression of all bricks of one time step takes 3.0 seconds. This is slightly faster than the upload of the uncompressed data, which takes 3.2 seconds (at a throughput of 4.2 GB/s over PCI-E). It has to be mentioned, however, that the decompression on the GPU blocks the GPU so that no other tasks can be performed. When uploading uncompressed data, the data transfer could be performed in parallel with other GPU tasks, such as rendering. Thus, operations on uncompressed data are usually slightly faster than the operations on the compressed data, but only if the data is already available in CPU memory. This can not be assumed in general, e.g., when stepping through multiple time steps, or when multiple (e.g. filtered) volumes are required simultaneously (see Sections 3.2 and 4.3). Whenever disk access becomes necessary, working on the compressed data becomes significantly faster. Reading a single compressed time step from disk takes only about 4.2 s. Additionally, reading can be performed concurrently with decompression and rendering, and, thus, it can usually be hidden completely. In contrast, reading an uncompressed time step from disk takes about 2.3 minutes.

Our display rates can not be compared to those reported for ray-casting of large scalar fields [6, 14], even though volume ray-casting on the bricked velocity field representation is used. This is because a) classical volume ray-casting systems typically make use of level-of-detail rendering, which is not admissible in our application, and b) exploit empty space skipping, which has no effect in our scenario where the data sets do not contain empty space. It is also worth mentioning that much more complex shaders are executed by our system for feature reconstruction. In our case, high-quality visualization using on-the-fly feature extraction and tri-cubic interpolation takes about 15.1 seconds, of which only 3 seconds are required for decompressing the velocity field and 12.1 seconds are required for ray-casting. The

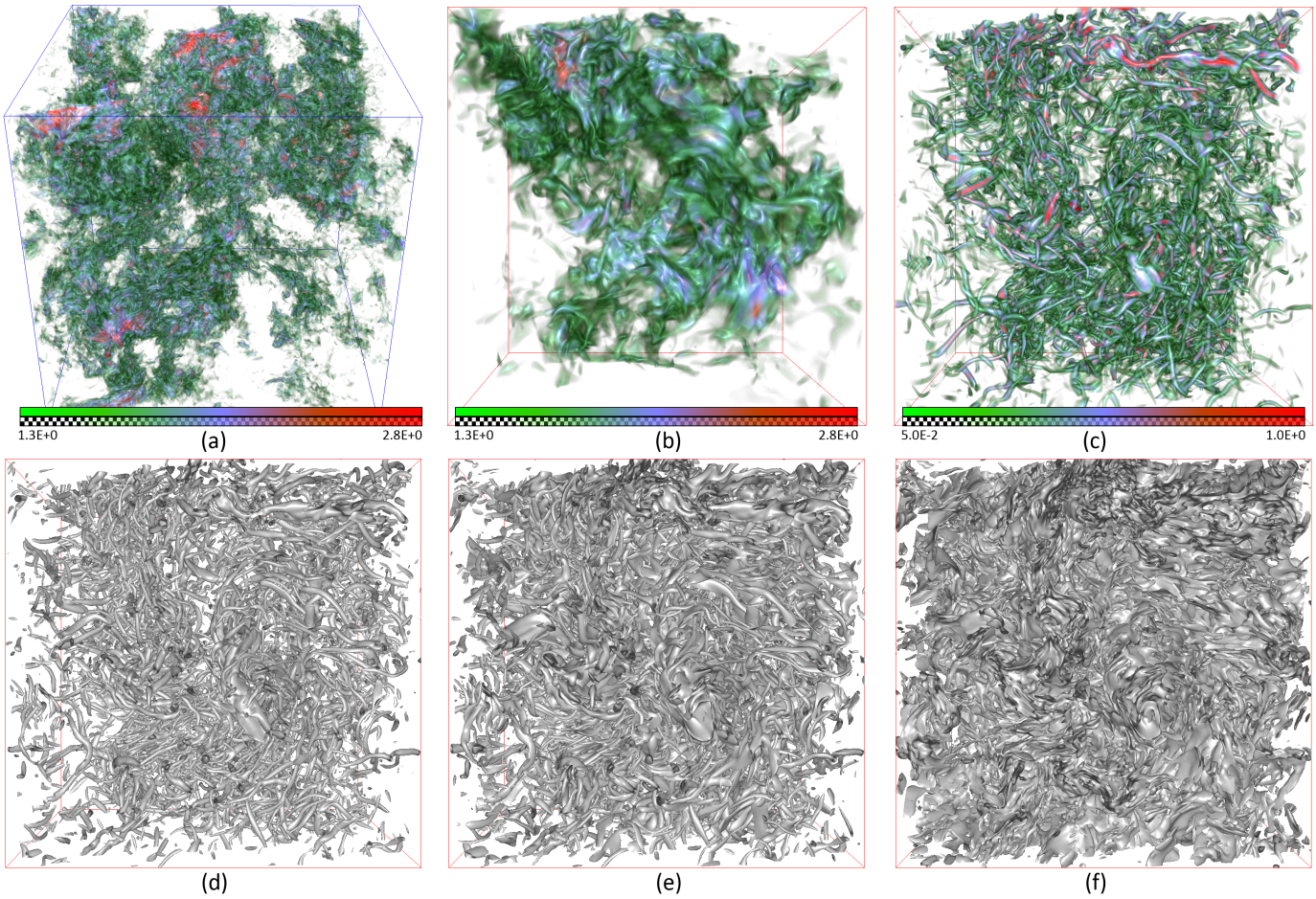

Fig. 15. Visualizations of forced isotropic turbulence. (a) Direct volume rendering of the velocity magnitude in the whole data set. Images (b-f) show a closeup on a 256^3 subregion at the center of the simulation domain. (b) Velocity magnitude. (c) Vorticity magnitude. Images (d-f) show iso-surfaces of invariants of the velocity gradient tensor (from left to right: Q_{Hunt}, Q_Ω, Q_S). In all three images, iso-surfaces of a value equal to 1.0E-2 are shown.

reason lies in the extremely complex shaders on the velocity gradient tensor, which are evaluated at every sample point to evaluate the selected feature metrics. As an alternative, a preview-quality visualization of a precomputed scalar feature volume using tri-linear interpolation takes about 1.5 seconds. The computation and compression of a scalar feature volume from a compressed vector field requires about 9.3 seconds. If the whole feature volume can be stored on the GPU, e.g. on a Quadro or Tesla card with 4-6 GB of video memory, rendering takes only 0.2 seconds. Since in this case the feature volume does not need to be compressed, it can be generated on the GPU in about 5.1 seconds.

A very expensive operation is filtering of the 3D velocity field. This operation requires the entire data set to be 3 times uploaded to the GPU, decompressed, filtered along one dimension, and compressed. Even though the raw compute time to filter the data on the GPU is only 0.3 seconds per filtering pass for a filter with a support of 51, it takes about 40 seconds until the result is available in CPU memory. This time is vastly dominated by the GPU decompression, and in particular the GPU compression of the intermediate and final results. Compression is about 3 times as expensive as decompression because the run-length and Huffman encoders are more complex than the respective decoders [39]. In particular, Huffman encoding requires a round-trip to the CPU, where the Huffman table is constructed. Table 2 summarizes the startup and per-frame time required for some common scenarios as outlined above. These times do not include the times required to load the data from disk, because disk access is performed concurrently with rendering and processing and can usually be hidden.

From the measured timing statistics it becomes clear that for the given data sets our system can not achieve fully interactive display rates. The reason is that the system was developed with the intent of visualizing extremely large turbulence data sets, which require complex shaders for feature extraction including high-quality interpolation. This is necessary to provide the significant amounts of fine detail at the smallest scale. However, the memory-efficient design of our system makes it well suited for implementation on desktop machines.

6 Conclusion

In this paper, we have presented a system for the exploration of very large and time-dependent turbulence data on desktop PCs. The interactive exploration of terascale data with limited available memory and bandwidth is made possible by the integration of a wavelet-based compression scheme. A fast GPU implementation of both compression and decompression provides the necessary throughput for fast streaming of velocity data and the efficient storage of derived data. We have demonstrated that the very high quality of the compression ensures the faithful preservation of turbulence features. A preview mode in the renderer based on precomputed feature volumes allows the interactive navigation to features of interest at only slightly reduced quality. On the other hand, the high-quality rendering of time-dependent image sequences is accelerated by an order of magnitude compared to the use of uncompressed data.

By using our system, a turbulence researcher can interactively explore terascale data sets and tune visualization parameters. While it is still too early to report on application-specific results, a first trend has already been discovered with the help of our system which had not been observed before: In multi-scale visualizations such as Fig. 12, the large-scale vortices contain small-scale vortices that appear to form

helical bundles within the large-scale vortices. They appear to wrap around the large-scale ones. These visualizations suggest new statistical measures such as the alignment angle between large- and small-scale vorticity, to be implemented in future research as a result of the present observations.

While our current system is tailored for desktop PC systems, we believe that many of the presented techniques also have applications in supercomputing. When moving to the petascale, computing will enable numerical simulations at unprecedented resolution and complexity, going beyond even the present turbulence data sets. Although the raw compute power of separate visualization computers keeps pace with those of supercomputers, bandwidth and memory issues in networking and file storage significantly restrict the reachable limits. For visualizing the peta- or even exabytes of data we will be confronted with, writing the raw data to disk or moving them across the network has to be avoided. A promising direction for future research is the integration of a compression layer similar to the one used in our system, which could alleviate bandwidth limitations between compute nodes and the visualization system. This could allow the immediate visualization of data too large even to be stored on disk.

Acknowledgments

This publication is based on work supported by Award No. UK-C0020, made by King Abdullah University of Science and Technology (KAUST).

References

[1] W. Ashurst, R. Kerstein, R. Kerr, and C. Gibson. Alignment of vorticity and scalar gradient with the strain rate in simulated Navier-Stokes turbulence. *Physics of Fluids*, 30:2343–2353, 1987. 4

[2] M. Burtscher and P. Ratanaworabhan. FPC: A high-speed compressor for double-precision floating-point data. *IEEE T. Computers*, 58(1):18–31, 2009. 2, 5

[3] B. J. Cantwell. Exact solution of a restricted Euler equation for the velocity gradient tensor. *Physics of Fluids*, A 4:782–793, 1992. 4

[4] M. S. Chong, A. E. Perry, and B. J. Cantwell. A general classification of three-dimensional flow fields. *Physics of Fluids*, 2:765–777, 1990. 4

[5] A. Cohen, I. Daubechies, and J. C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Comm. Pure and Applied Mathematics*, 45(5):485–560, 1992. 3

[6] C. Crassin, F. Neyret, S. Lefebvre, and E. Eisemann. GigaVoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *ACM SIGGRAPH Interactive 3D Graphics and Games (I3D)*, 2009. 2, 7

[7] D. Ellsworth, B. Green, and P. Moran. Interactive terascale particle visualization. In *IEEE Visualization*, pages 353–360, 2004. 2

[8] T. Fogal, H. Childs, S. Shankar, J. Krüger, R. D. Bergeron, and P. Hatcher. Large data visualization on distributed memory multi-GPU clusters. In *High Performance Graphics (HPG)*, pages 57–66, 2010. 2

[9] N. Fout, H. Akiba, K.-L. Ma, A. E. Lefohn, and J. Kniss. High-quality rendering of compressed volume data formats. In *EG/IEEE VGTC Visualization (EuroVis)*, pages 77–84, 2005. 3

[10] N. Fout, K.-L. Ma, and J. Ahrens. Time-varying, multivariate volume data reduction. In *ACM Applied Computing*, pages 1224–1230, 2005. 3

[11] J. E. Fowler and R. Yagel. Lossless compression of volume data. In *Volume Visualization (VVS)*, pages 43–50, 1994. 2

[12] R. Fraedrich, M. Bauer, and M. Stamminger. Sequential data compression of very large data in volume rendering. In *Vision, Modeling and Visualization (VMV)*, pages 41–50, 2007. 2

[13] U. Frisch. *Turbulence, the legacy of A.N. Kolmogorov*. 1995. 4

[14] E. Gobbetti, F. Marton, and J. Iglesias Guitián. A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *The Visual Computer*, 24(7-9):797–806, 2008. 2, 7

[15] S. Guthe and W. Straßer. Real-time decompression and visualization of animated volume data. In *IEEE Visualization*, pages 349–356, 2001. 3, 6

[16] S. Guthe, M. Wand, J. Gonser, and W. Straßer. Interactive rendering of large volume data sets. In *IEEE Visualization*, pages 53–60, 2002. 3, 5

[17] G. Haller. An objective definition of a vortex. *J. Fluid Mechanics*, 525:1–26, 2005. 4

[18] M. Howison, E. W. Bethel, and H. Childs. MPI-hybrid parallelism for volume rendering on large, multi-core systems. In *EG Parallel Graphics and Visualization (EGPGV)*, 2010. 2

[19] J. Hunt, A. Wray, and P. Moin. Eddies, streams, and convergence zones in turbulent flows. *Centre for Turbulence Research, CTR-S88*, 1988. 4

[20] T. Ishihara, T. Gotoh, and Y. Kaneda. Study of high-Reynolds number isotropic turbulence by direct numerical simulation. *Annu. Rev. Fluid Mechanics*, 41:165–180, 2009. 1

[21] J. Jeong and F. Hussain. On the identification of a vortex. *J. Fluid Mechanics*, 285:69–94, 1995. 4

[22] J. Katz and J. Sheng. Applications of holography in fluid mechanics and particle dynamics. *Annu. Rev. Fluid Mechanics*, 42:531–555, 2010. 1

[23] M. Kraus. Scale-invariant volume rendering. In *IEEE Visualization*, pages 295–302, 2005. 3

[24] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *IEEE Visualization*, pages 287–292, 2003. 3

[25] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. Samatova. Compressing the incompressible with ISABELA: In-situ reduction of spatio-temporal data. In *Parallel and Distributed Computing (Euro-Par)*, pages 366–379, 2011. 3

[26] Y. Li, E. Perlman, M. Wan, Y. Yang, C. Meneveau, R. Burns, S. Chen, A. Szalay, and G. Eyink. A public turbulence database cluster and applications to study Lagrangian evolution of velocity increments in turbulence. *J. Turbulence*, 9:N31, 2008. 2

[27] P. Lindstrom and M. Isenburg. Fast and efficient compression of floating-point data. *IEEE T. Visualization and Computer Graphics (Proc. Visualization)*, 12(5):1245–1250, 2006. 2, 5

[28] E. Lum, K.-L. Ma, and J. Clyne. Texture hardware assisted rendering of time-varying volume data. In *IEEE Visualization*, pages 263–270, 2001. 3

[29] J. Martin, A. Ooi, M. S. Chong, and J. Soria. Dynamics of the velocity gradient tensor invariants in isotropic turbulence. *Physics of Fluids*, 10:2336–2346, 1998. 4

[30] B. Moloney, M. Ament, D. Weiskopf, and T. Möller. Sort-first parallel volume rendering. *IEEE T. Visualization and Computer Graphics*, 17(8):1164–1177, 2011. 2

[31] D. Nagayasu, F. Ino, and K. Hagihara. Two-stage compression for fast volume rendering of time-varying scalar data. In *Computer Graphics and Interactive Techniques (GRAPHITE)*, pages 275–284, 2006. 2

[32] D. Nagayasu, F. Ino, and K. Hagihara. A decompression pipeline for accelerating out-of-core volume rendering of time-varying data. *Computers & Graphics*, 32(3):350–362, 2008. 2

[33] K. G. Nguyen and D. Saupe. Rapid high quality compression of volume data for visualization. *Computer Graphics Forum*, 20(3):49–57, 2001. 3, 5

[34] K. Sayood. *Introduction to Data Compression*. 3rd edition, 2005. 2, 5

[35] J. Schneider and R. Westermann. Compression domain volume rendering. In *IEEE Visualization*, pages 293 –300, 2003. 3

[36] C. Sigg and M. Hadwiger. Fast third-order texture filtering. In *GPU Gems 2*, pages 313–329. 2005. 3

[37] K. Sreenivasan and R. Antonia. The phenomenology of small-scale turbulence. *Annu. Rev. Fluid Mechanics*, 29:435–472, 1997. 4

[38] D. S. Taubman and M. W. Marcellin. *JPEG 2000: Image Compression Fundamentals, Standards and Practice*. 2001. 5

[39] M. Treib, F. Reichl, S. Auer, and R. Westermann. Interactive editing of gigasample terrain fields. *Computer Graphics Forum (Proc. Eurographics)*, 31(2):383–392, 2012. 3, 5, 8

[40] W. J. van der Laan, A. C. Jalba, and J. B. T. M. Roerdink. Accelerating wavelet lifting on graphics hardware using CUDA. *IEEE T. Parallel and Distributed Systems*, 22(1):132–146, 2011. 3

[41] J. S. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33(2):209–271, 2001. 2

[42] J. Wallace and P. Vukoslavcevic. Measurement of the velocity gradient tensor in turbulent flows. *Annu. Rev. Fluid Mechanics*, 42:157–181, 2010. 1

[43] R. Westermann. A multiresolution framework for volume rendering. In *Volume Visualization (VVS)*, pages 51–58, 1994. 3, 5

[44] J. Woodring, S. Mniszewski, C. Brislawn, D. DeMarle, and J. Ahrens. Revisiting wavelet compression for large-scale climate data using JPEG 2000 and ensuring data precision. In *IEEE Large-Scale Data Analysis and Visualization (LDAV)*, pages 31–38, 2011. 3

[45] B.-L. Yeo and B. Liu. Volume rendering of DCT-based compressed 3D scalar data. *IEEE T. Visualization and Computer Graphics*, 1(1):29–43, 1995. 3, 5