DOI: 10.1111/cgf.70471

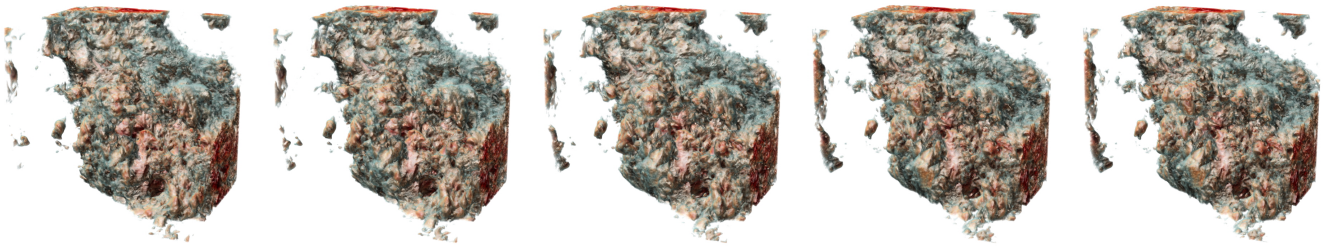
COMPUTER GRAPHICS *forum*

Eurographics Conference on Visualization (EuroVis) 2026
N. Andrienko, M. Chen, and B. Wang
(Guest Editors)

TT4D: Tensor-Train-based 4D Time-Dependent Volume Rendering

Clara Hartmann[1] Rafael Ballester-Ripoll[2] Renato Pajarola[1]

[1]University of Zurich, Department of Informatics, Switzerland
[2]IE University, Madrid, Spain



(a) $t = 1$, *full resolution, reconstruction time* = 70*ms* **(b)** $t = 21$, $1:8$ *subsampled, reconstruction time* = 11*ms* **(c)** $t = 42$, $1:8$ *subsampled, reconstruction time* = 9*ms* **(d)** $t = 63$, $1:8$ *subsampled, reconstruction time* = 10*ms* **(e)** $t = 63$, *full resolution, reconstruction time* = 37*ms*

Figure 1: *Time steps t from a 4D time-dependent turbulence dataset of size* $(512^3 \times 64)$*, compressed with a ratio of* $1:290$*. Images (a) and (e) are reconstructed in full resolution; (b), (c), and (d) are rendered at* $1:8$ *subsampling to interactively* move *through time.*

Abstract

Visualizing large-scale, time-varying volumetric data using direct volume rendering remains a significant challenge in scientific visualization, particularly as data sizes and resolutions continue to increase. One promising direction for addressing this challenge is the use of tensor decompositions, which have proven to be a useful tool for the compact representation of large, high-dimensional data. However, their integration into time-dependent interactive visualization pipelines remains limited. We introduce TT4D*, a memory- and time-efficient lossy compression and decompression technique for four-dimensional (4D) time-varying volume data based on the tensor train (TT) decomposition. Our approach employs efficient subsampling during decomposition to reduce memory consumption and computational cost, enabling the processing of large datasets while avoiding unnecessarily large intermediate matrices and tensors. Beyond compression, TT4D provides a GPU-based, on-the-fly decoding scheme that avoids reconstructing the entire 4D volume, supporting interactive visualization. At equivalent error levels, TT4D outperforms other transform-based compressors at random-access decompression across datasets up to 64GB. Exploiting the structure of TT cores, our approach enables adaptive multiresolution rendering, fast spatio-temporal data exploration, and interactive filtering. Together, these capabilities make TT4D a scalable solution for interactive visualization of high-resolution, time-varying volume data.*

Keywords: volume visualization, tensor approximation, compression domain volume rendering, time-varying volume data, interactive visualization

1. Introduction

Four-dimensional (4D) time-varying volume rendering has a multitude of applications, such as in bio-medical imaging, computational fluid dynamics, or weather simulations. The growing precision of sensors and greater computational power have led to higher data resolutions, resulting in larger datasets. The large sizes of scientific datasets make it hard to store, render, and explore them with filters or data cutouts, which is often needed for interactive visualization and analysis of scientific datasets. One way to visualize 3D datasets is direct volume rendering (DVR). DVR allows standard visual exploration of volumetric datasets. For time-dependent data, a DVR tool should be interactive, allowing the user to inspect spatial regions and to explore the different time steps interactively. Note that the added time dimension leads to an even larger data space compared to 3D data, as one 3D volume needs to be stored for every time step individually. With potentially large numbers of time steps and the need for fast random access to individual time steps, the interactive visualization of 4D time-varying volumes be-

comes particularly challenging. These problems often lead to restrictions in data size, which are commonly addressed by data compression techniques. These, in turn, lead to long compression times or slow data access, e.g., due to expensive decompression, which makes real-time interactions difficult. In addition to standard visualization, the use of multiresolution rendering techniques or the application of filter operations are common desirable functionalities to visualize large data efficiently, avoiding spending excessive time on rendering the full-resolution data at each point in time.

As one useful tool for data compression, tensor decompositions have shown considerable effectiveness across a range of application domains [WXC*08, WWS*05, RK09, IAVHDL11, BRP16a, ABK16, BRP19, BRLP20]. Many simulation or scientific datasets show a high linear correlation over time and spatial dimensions. This property makes them well compressible with factorization techniques such as tensor decompositions. Using the factorization of large, high-dimensional data into multiple tensors of smaller size and dimension, the dataset can be stored memory-efficiently while preserving high accuracy. Tensor train (TT) decompositions [Ose11] are an especially interesting approach for the compression of higher-dimensional data, as their cores have a restricted maximum order of 3, leading to small compressed data sizes and linear growth with increasing dimensionality of the data. However, the construction of a TT decomposition of large 4D (or higher-dimensional) datasets is a complex task, requiring several costly operations. These operations include multiple singular value decompositions (SVD) on large unfolded, imbalanced matricized tensors, which require time and the creation of large intermediate matrices, and multiple tensor reshaping operations on large intermediate matrices. This leads to significant memory consumption as well as excessive memory access to many data elements during the TT decomposition process. In addition to the compression, the visualization of the compressed data is also costly, requiring the reconstruction of the compressed data. The reconstruction requires multiple tensor-tensor multiplications to decode the tensor decomposition. For rendering 4D time-varying volume data, this reconstruction is prohibitively slow when accessing different time steps, especially for access in a non-sequential and sometimes rapidly changing manner. This makes any standard implementation of tensor reconstruction unusable for interactive visualization of large time-varying volume data.

These challenges have motivated us to introduce a novel memory-efficient and time-effective TT decomposition approach, as well as an interactive GPU-based decoder and renderer for TT-based 4D volume data. Our decomposition approach is based on the sequential SVD (SSVD) TT decomposition method [Ose11], but uses an efficient subsampling method to avoid full-scale tensor reshaping and SVD computations on large imbalanced matrices to speed up the computation and reduce memory consumption. Additionally, our decoder uses an efficient partial preprocessing of the compressed data, which reduces the computational load for decoding (arbitrary) time steps interactively on the GPU. Furthermore, we have integrated an adaptive multiresolution reconstruction strategy to speed up the rendering during heavy user interactions by decoding the data partially as needed. Unlike most non-tensor-based compression techniques, the linear nature of the TT cores further allows for the application of quick spatio-temporal data selection and efficient filtering operations in the compressed data domain. We demonstrate and test our *TT4D* approach on different time-dependent real-world volume data sets of different sizes up to 64GB, offering real-time rendering capabilities with interactive filtering and data selection. TT4D is designed as an interaction-enabling representation for large time-varying volume data, prioritizing real-time exploration, random access, and scalability over highest-accuracy reconstruction.

2. Related Work

Volume compression A large variety of compression methods for scientific volume data have previously been proposed, covering a wide range of techniques such as wavelet or DCT based methods [Mur93, YL95, GEA96, GLDH97, KS99, IP99], transform and vector quantization based coding [NH92, FM07, Lin14], embedded coding techniques [Lin14], prediction error based coding [DC16], and tensor decomposition [BRP16a, BRLP20, BKK20]. Often, the operations involved (multilinear transformations, predictive coding, variable bit-length encoding, etc.) are tailored to maximize compression ratios. However, this may prevent their direct use in interactive on-the-fly decoding and rendering application scenarios where only a few milliseconds per frame are available, and are rarely amenable to efficient slicing or random-access patterns. Methods that do support random access (such as, e.g., the fixed-rate modality in the ZFP [Lin14] floating-point compressor) do so at the expense of less attractive compression ratios.

In the area of tensor approaches, a Tucker decomposition based compression method TTHRESH [BRLP20] was shown to be competitive in a number of aspects compared to other established methods. Other tensor-based methods include Tucker truncation [BRP16a] and TuckerMPI [BKK20], the latter being able to compress very large data with high compression ratios using supercomputers. These algorithms show promising compression ratios and accuracy, making them attractive for storage and data transmission scenarios, but they are not optimized for real-time slicing and rendering, especially in the time-varying case. In contrast and to allow fast decoding as well as to support compression domain data selection and processing operations, our approach guarantees fixed-rate compression (and, therefore, random-access patterns) by avoiding compression techniques such as variable-bit length encoding or vector quantization.

Multiresolution volume rendering Real-time GPU-based DVR of large-scale volume data using variable resolution level-of-detail techniques has been extensively explored in previous work, see also [BHP15], and has become a de facto standard. SparseLeap [HAAB*18], for instance, exploits intelligent skipping of empty regions to accelerate rendering. The challenge of rendering multiresolution compressed volume data has also been addressed and reviewed in the literature [BRGI*14, BRGI*13]. Besides GPU acceleration, many of these approaches include variable-length or sparsely encoded volumes [LK11, GG16, SKMH14] as well as tensor-based methods [BRSP15, BRSP18], which are most closely related to our proposed approach.

Unlike most other compression methods, representations of large

multidimensional data sets based on tensor decomposition additionally permit complex operations to be performed directly in the compressed domain [BRP16c], such as spatial transformations and selection [SMP13], filter operations [BRSP18], multidimensional integrals [BRP19], or structural in-painting [BRP16b].

4D volume rendering Compressed volume rendering methods that work well on 3D volume data may fall short for 4D time-varying volumes. This is because of the need for continuous and possibly non-sequential reconstruction of, and (random) access to individual time steps at interactive frame rates. Various interactive 4D time-dependent volume rendering methods have been proposed, including texture and image-based approaches [LMC01, BTL20, RPD22]. A fast GPU-accelerated out-of-core 4D volume renderer has been proposed in [MAG19] that exploits a novel low-bitrate codec. Other methods achieve fast rendering speeds at the expense of high compression times, e.g., using quantization techniques [WYM10, YZW*17]. Similarly, techniques based on neural representations show very high compression ratios and fast rendering [LJLB21, WHW22]. In contrast to tensor approximation methods, however, these approaches generally do not allow for interactive compression domain data processing or filtering operations.

To tackle these limitations, we propose a novel method consisting of a memory- and time-efficient compression step, which compresses the given 4D volumetric data using TT decompositions, as well as a fast GPU-based decoding and interactive rendering. In particular, our method contains an intelligent GPU-based reconstruction approach that allows for real-time reconstruction of arbitrary time steps and partial data. Reconstruction is fully based on multilinear transformations (matrix and tensor products), therefore slicing or linear filtering operations can be efficiently applied to the data at interactive rates without decompressing the complete 4D dataset. This allows us to render data originally larger than the available GPU memory. We will provide experimental results showing a detailed comparison of our TT4D approach to other best-of-class compression methods, demonstrating their limitations.

3. Method

Compression and reconstruction of 4D data can be very time-consuming and memory-intensive if not adapted for a certain use case. Our proposed TT4D method consists of two main parts. First, we introduce a memory- and time-efficient compression method based on the tensor train (TT) decomposition. Second, we introduce an efficient on-the-fly adaptive reconstruction and rendering approach for the interactive visualization of individual time steps of time-varying volume data, supporting random as well as continuous time access.

3.1. 4D Tensor Train Construction

Our data compression method is based on the TT decomposition [Ose11], which factorizes an input tensor into a set of two- and three-dimensional tensors, called the *TT cores* (see Figure 2). For a tensor $\mathscr{T} \in \mathbb{R}^{N_1 \times \ldots \times N_d}$, the TT is given by

$$\mathscr{T} = \sum_{r_1} \cdots \sum_{r_{d-1}} \mathscr{C}_1[:, r_1] \circ \mathscr{C}_2[r_1, :, r_2] \circ \cdots \circ \mathscr{C}_d[r_{d-1}, :], \quad (1)$$

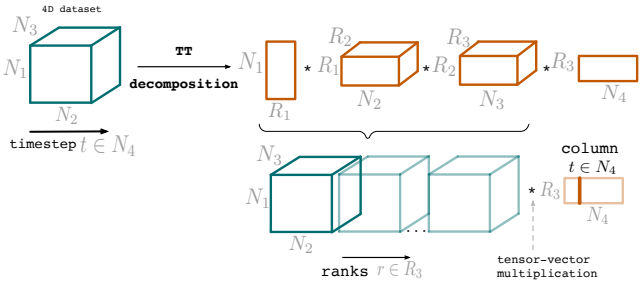


Figure 2: *TT decomposition for a 4D dataset, plus scheme of reconstruction. Reconstruction is preprocessed by multiplying the first 3 cores. Reconstruction of a time step t is done by multiplication of the preprocessed tensor with the t-th column vector of the last core.*

with TT cores $\mathscr{C}_{i=1\ldots d} \in \mathbb{R}^{R_{i-1} \times N_i \times R_i}$, $R_0 = R_d = 1$ and "$\circ$" denoting the outer product [Ose11, LLLZ22]. In contrast to other tensor decompositions, like the Tucker model, the dimensionality of the TT cores does not grow with the dimensionality of the original tensor. Thus, larger 4D volumes do not cause a quartic but only a linear growth in the TT representation, making it especially suitable for high-dimensional data. For more details on TT as well as basic tensor notations, see [Ose11] and [LLLZ22].

The naive implementation of the computation of the TT decomposition is based on the sequential singular value decomposition (SSVD), also called the Tensor-Train Singular Value Decomposition (TT-SVD) [Ose11]. However, SSVD is impractical for processing large data since it requires repeatedly unfolding and reshaping the full 4D tensor, which is extremely memory- and time-intensive. In addition, the method involves the computation of multiple SVDs on very large unfolded matrices, some as large as the entire input tensor. The unfolded matrices are additionally highly imbalanced, having far more columns than rows. These challenges have motivated us to introduce a more efficient variant of the SSVD.

3.1.1. Efficient TT Decomposition

To overcome the problems of the SSVD, we propose a novel approach that allows us to work only on parts of the full data. Figure 3 illustrates the complete process of our proposed memory-optimized decomposition approach. Let $\mathscr{T}$ be the 4D input tensor of size $N_1 \times N_2 \times N_3 \times N_4$ and R_1, R_2, R_3 our predefined ranks of the TT decomposition. In the original SSVD, the first step is a mode-1-unfolding (see Figure 4) of the tensor into a (very wide) matrix $\mathbf{A} \in \mathbb{R}^{N_1 \times (N_2 \cdot N_3 \cdot N_4)}$. The first dimension of $\mathbf{A}$ is the first *mode* of the tensor, and the second dimension is the product of all remaining modes. The next step is to compute a factorization $\mathbf{A} \approx \mathbf{U} \cdot \mathbf{V}^\intercal$ using SVD, where $\mathbf{U} \in \mathbb{R}^{N_1 \times R_1}$, given the first rank R_1, represents the first R_1 left-singular vectors of $\mathbf{A}$, and $\mathbf{V}^\intercal$ the right-singular vectors multiplied with the singular values.

Not only is the SVD on such a large matrix memory- and time-consuming, but we also observe that $\mathbf{A}$ is highly imbalanced, leading to the fact that most of the columns are linearly dependent. We note that the linear column space of $\mathbf{A}$ is spanned by at most $N_1 \ll (N_2 \cdot N_3 \cdot N_4)$ linearly independent columns, making all other $(N_2 \cdot N_3 \cdot N_4) - N_1$ columns linearly dependent. Thus, it is suffi-

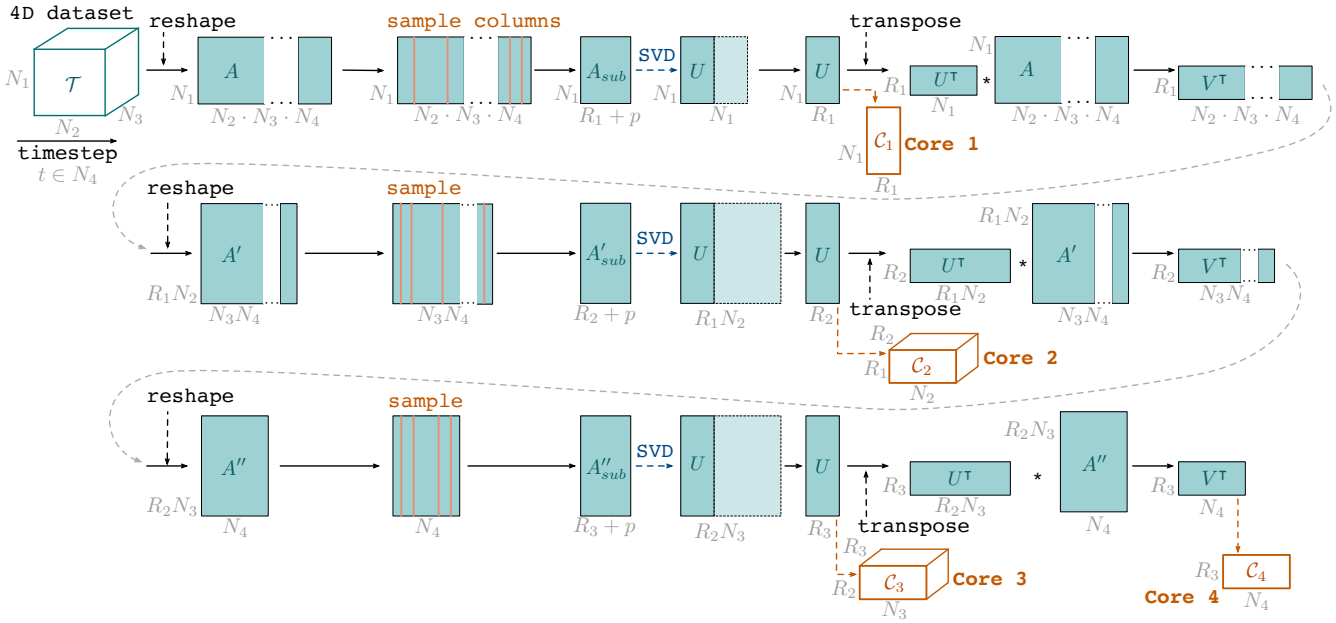


Figure 3: *Memory-efficient sequential singular value decomposition to obtain a TT from a 4D input tensor of shape* $(N_1 \times N_2 \times N_3 \times N_4)$ *with given ranks* (R_1, R_2, R_3) *using column subsampling.*

cient to use only N_1 linearly independent columns of $\mathbf{A}$ to find the left singular vectors forming $\mathbf{U}$. Moreover, we are only interested in the first $R_1 < N_1$ most significant left singular vectors, which can be computed from a subset of linearly independent columns $\mathbf{A}_{sub}$ [OT10]. Based on this observation, we propose an adapted, improved version of the SSVD-based TT decomposition.

As shown in [CW19], it is possible to compute the left-singular vectors of $\mathbf{A}$ on $R_1 + p$ vectors, with an oversampling parameter p. Each of the $R_1 + p$ vectors is defined as a linear combination of the columns of $\mathbf{A}$ using random Gaussian weights. This approach has the advantage of a computable error bound for the resulting approximation of $\mathbf{A}$. The downside of this method is that we still have to access all $N_1 \cdot N_2 \cdot N_3 \cdot N_4$ elements in the unfolded tensor $\mathscr{T}$, which is too costly for large 4D tensors.

Instead of computing the $R_1 + p$ columns from linear interpolations of all columns as in [CW19], we propose an adapted *random sampling* algorithm for selecting a subset of $R_1 + p$ columns of $\mathbf{A}$ forming $\mathbf{A}_{sub}$. We will discuss different selection techniques below in Section 3.1.2. Using the sampling algorithm, we can now compute $\mathbf{U}$ using SVD on a much smaller submatrix $\mathbf{A}_{sub}$. $\mathbf{U}$ then defines the first TT core $\mathscr{C}_1$.

For the next step of the SSVD TT construction, we need the right factor of $\mathbf{A} = \mathbf{U} \cdot \mathbf{V}^\intercal$. As we compute the SVD only from $\mathbf{A}_{sub}$ with a reduced number of columns, the linear space of the right singular vectors of $\mathbf{A}$ is not spanned properly. That is, we cannot directly use the right singular vectors of the SVD of $\mathbf{A}_{sub}$. Instead, we reconstruct the correct right singular vectors $\mathbf{V}^\intercal$ as follows. Let

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{V}^\intercal \tag{2}$$

denote the factorization with $\mathbf{V}^\intercal$ being the matrix to compute. By definition, $\mathbf{U}$ is an orthonormal matrix, hence $\mathbf{U}^\intercal \cdot \mathbf{U} = \mathbf{I}$ with $\mathbf{I}$ being the identity matrix. It follows that

$$\mathbf{U}^\intercal \cdot \mathbf{A} = \mathbf{U}^\intercal \cdot \mathbf{U} \cdot \mathbf{V}^\intercal \Rightarrow \mathbf{V}^\intercal = \mathbf{U}^\intercal \cdot \mathbf{A}. \tag{3}$$

Hence, we can compute the required matrix $\mathbf{V}^\intercal$ from the product of the transposed left singular vectors $\mathbf{U}^\intercal$ and $\mathbf{A}$. This avoids the expensive SVD computation on the full matrix $\mathbf{A}$.

$\mathbf{V}^\intercal \in \mathbb{R}^{R_1 \times (N_2 \cdot N_3 \cdot N_4)}$ builds the basis for the next step of SSVD. Therefore, it needs to be reshaped into $\mathbf{A}'$ such that

$$\text{reshape}(\mathbf{V}^\intercal) =: \mathbf{A}' \in \mathbb{R}^{(R_1 \cdot N_2) \times (N_3 \cdot N_4)}. \tag{4}$$

However, this reshaping operation can be expensive in terms of memory and time if it requires an explicit memory copy of the large matrix $\mathbf{V}^\intercal$. In our approach, we avoid this overhead by exploiting column-major storage, which enables implicit reshaping. Given a matrix in column-major format, the columns are stored contiguously in memory. Hence, reshaping the matrix from shape $\mathbb{R}^{R_1 \times (N_2 \cdot N_3 \cdot N_4)}$ to shape $\mathbb{R}^{(R_1 \cdot N_2) \times (N_3 \cdot N_4)}$, as illustrated in Figure 4, merely concatenates sequential columns into longer columns. Consequently, the reshaping reduces to an index reinterpretation, requiring no explicit memory access ot data movement.

The previous procedure is then repeated iteratively with $\mathbf{A}'$, where in each step the left singular vectors are reshaped and stored as a 3D TT core and the right singular vectors serve as a starting point for the next iteration, leading to a new $\mathbf{A}'$. We continue until we have obtained three TT cores like that, and the fourth and last TT core is then defined as the last matrix $\mathbf{V}^\intercal$ of the last factorization. The TT cores $\mathscr{C}_{i=1\ldots4}$ have shapes $\mathbb{R}^{R_{i-1} \times N_i \times R_i}$, respectively, with $R_0 = R_4 = 1$. Hence, the first and last TT cores are matrices, while the two inner TT cores are 3D tensors.

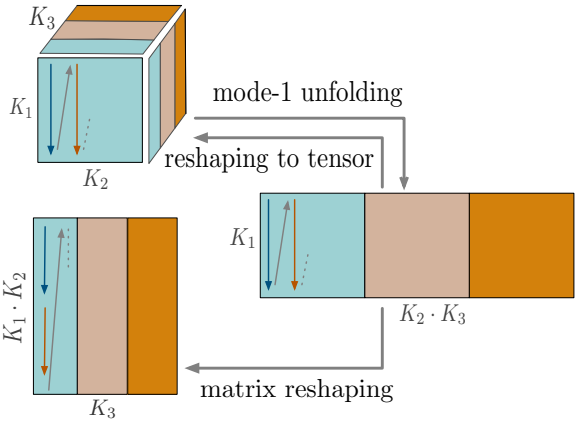


Figure 4: *mode-1 unfolding of a tensor of size* $(K_1 \times K_2 \times K_3)$ *and reshaping of a* $(K_1 \times (K_2 \cdot K_3))$*-matrix.*

3.1.2. Column Sampling

As outlined above, the computation of the SVD on large matrices is expensive, but wide, imbalanced matrices have their column space highly over-defined. This allows us to compute the left singular vectors from a subset of columns, as long as the set is chosen wisely. We present two efficient methods to select such a submatrix, the first using completely random column selection and the second using the maximum volume approximation approach [MO15].

Random sampling is the more efficient and simpler approach of the two proposed approaches. In each iteration, let $\mathbf{A} \in \mathbb{R}^{M \times N}$ with $M \ll N$ be the matrix of which we want to obtain the first R_i left singular vectors by applying SVD. We propose to randomly select $R_i + p$ of the columns of $\mathbf{A}$ which define the submatrix $\mathbf{A}_{sub}$, where $p \geq 0$ is a predefined oversampling parameter. The first R_i left singular vectors can be computed using SVD on the submatrix $\mathbf{A}_{sub}$. This works in most cases because the chance that R_i of the selected $R_i + p$ columns are linearly independent is very high. This is because with $p > 0$, we choose more columns than needed, of which only R_i need to be linearly independent. When computing the SVD, we can automatically check whether the input matrix has full rank. If not, we know that the selected set does not contain R_i linearly independent vectors, and we select a new set of random columns.

While random column sampling is very time- and memory-efficient, it does not guarantee the best possible set of columns. Hence, we might find a set of left singular vectors that do not necessarily represent our matrix well. Therefore, we introduce a second approach. It is based on the maximum volume approximation (maxvol) approach [MO15]. The idea is based on the fact that the best choice of $R_i + p$ vectors is defined by spanning a volume larger than any other set of $R_i + p$ chosen columns. Therefore, we try to improve the choice of columns iteratively, alternating between choosing optimal columns and optimal rows. Furthermore, we apply random preselection before applying the maxvol algorithm. That is because for an input matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$ with $M < N$, maxvol computes a matrix of size $N \times N$ as an intermediate step. As in our case, the dimension N can become quite large for large input matrices, we propose to randomly subsample the columns or rows, respectively, before applying the maximum volume algorithm. The complete process is shown in Algorithm 1.

Algorithm 1 Calculate a submatrix of a given matrix using alternating maximum volume approximation.

Input: Matrix $\mathbf{A} \in \mathbb{R}^{M \times N}, M < N$, rank R, oversampling parameter p, oversampling factor F
Let $\mathbf{A}_{sub} := \mathbf{A}[:, I_c]$, for I_c being $(R+p)$ random, uniform column indices of $\mathbf{A}$.
for $i = 0$ **to** max_iter **do**
  \\ Sample rows
  $I \leftarrow (F \cdot (R+p))$ random row indices of $\mathbf{A}_{sub}$
  $\mathbf{A}'_{sub} \leftarrow \mathbf{A}_{sub}[I, :]$
  $I_r \leftarrow \text{maxvol}(\mathbf{A}'_{sub})$ \\ indices returned by maxvol
  Let $\mathbf{R} \leftarrow \mathbf{A}[I_r, :]$
  \\ Sample columns
  $I \leftarrow (F \cdot (R+p))$ random column indices of $\mathbf{R}$
  $\mathbf{R}' \leftarrow \mathbf{R}[:, I]$
  $I_c \leftarrow \text{maxvol}(\mathbf{R}')$ \\ indices returned by maxvol
  $\mathbf{A}_{sub} \leftarrow \mathbf{A}[:, I_c]$
end for
return $\mathbf{A}_{sub}$

3.2. Interactive Reconstruction and Rendering

After decomposing the large 4D volume $\mathcal{T}$ into TT format with cores $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4$, we need to decode them for visualization. We assume the visualization of a 3D volume is using DVR, showing one time step at a time, and providing a time slider for interactive choice of, or random access to, the time steps. Naively, we could reconstruct the complete 4D data before visualization through three tensor-tensor multiplications [Ose11,LLLZ22]. However, that would defeat the purpose of limiting memory usage, as the full 4D tensor would be recreated. Alternatively, one could reconstruct each required time step on demand. This reconstruction requires two tensor-tensor multiplications and one tensor-vector multiplication. The amount of multiplications needed for this approach makes it too slow for interactive time access, as soon as we have to deal with large datasets.

To enable fast interactive time access, we propose a more efficient on-the-fly partial decoding approach. In the following, we assume, w.l.o.g., that the first three dimensions of the input data refer to the spatial dimensions, while the last dimension defines the time. We can observe that all TT cores $\mathcal{C}_{1,2,3}$ but the last one $\mathcal{C}_4$ are independent of time. Hence, we propose a partial decoding step, in which the first three TT cores $\mathcal{C}_{1,2,3}$ are multiplied together. The result is stored as a reduced base tensor $\widehat{\mathcal{T}}$ of dimensions $N_1 \times N_2 \times N_3 \times R_3$, where R_3 is the third TT decomposition rank (see Figure 2). Even though $\widehat{\mathcal{T}}$ is still a 4D tensor, it is significantly smaller than the fully reconstructed 4D volume, since $R_3 \ll N_4$. This allows us to render data that can be up to multiple times larger than available memory, even without any asynchronous loading or out-of-core data management.

The partial decoding into $\widehat{\mathcal{T}}$ requires two tensor-tensor multiplications but only happens once. Furthermore, the multiplications of

the elements are linear operations, independent of each other per tensor multiplication. Thus, this can be parallelized and highly accelerated using GPU computation.

At runtime, for each required time step, we only need to multiply the 4D base tensor $\widehat{\mathscr{T}}$ with one column vector of the last TT core $\mathscr{C}_4$, requiring a single tensor-vector multiplication to obtain the 3D volume of a specific time step. As these multiplications are vector dot products, which are independent linear operations again, they can also be parallelized on the GPU. The reduced number of operations necessary per reconstructed time step and the GPU acceleration allow us to achieve an interactive, flexible reconstruction of any requested time step.

3.2.1. Multiresolution Volume Sampling

For large data, the above reconstruction strategy may achieve interactive rates but may not reach fully smooth and fluid rendering for quickly changing time steps. Therefore, we propose an adaptive and progressive spatial reconstruction scheme. We assume that during fast interactions with the time selector, the full resolution of the 3D volume is not ultimately necessary to give a dynamic overview of the time-varying data. Hence, during active sliding through time, an adaptive lower spatial resolution of the data is reconstructed, which is progressively increased to full resolution as soon as the time sliding slows down.

The progressive decoding partially reconstructs a subsampled 3D time-slice from the 4D base tensor $\widehat{\mathscr{T}}$ for rendering. Hence, only a fraction of the spatial volume resolution is initially reconstructed, and the remaining voxels are progressively decoded as soon as possible based on frame rendering times. Note that this subsampled reconstruction can be obtained efficiently by reading out and decoding only a fraction of the full base tensor, based on a spatial subsampling pattern. As tensor-vector multiplication, a spatial subtensor of $\widehat{\mathscr{T}}$ is multiplied with the respective time column vector of the last TT core $\mathscr{C}_4$, resulting in the values of a subset of voxels of the full data. These values are then copied to the non-computed voxels, which are not part of the subset. The subsampling only happens in the spatial dimensions.

Our progressive sampling is based on a repetitive cube scheme, dividing the full 3D volume into small cubes as indicated in Figure 5. A subsampling ratio of $1:8$ divides the volume into $2\times2\times2$ cubes and then decodes only one of the eight voxels per cube, and the other seven are mere copies. Progressively, we can easily increase the sampling to $1:4$ by decoding the cube's diagonal element. Other patterns and sampling ratios could also be defined, especially for larger cubes. The benefit of this approach over a more complex interpolation scheme is that we compute a subset of the final data values. Hence, when going from a lower resolution to a higher resolution, e.g., $1:8$ to $1:4$ and then $1:1$, all the previously computed values can be reused and do not have to be recomputed again.

Furthermore, the subsampling scheme is adaptive, and the targeted resolution is based on the speed of computation. Starting from a predefined resolution of, e.g., $1:8$, we adaptively refine the resolution of the current time step until the computation time exceeds $30ms$. Stopping the time slider eventually leads to the computation of the full resolution. This scheme allows for smooth real-time rendering of lower resolution versions of the volume, even for fast-changing time values, while still providing the full resolution for detailed analysis of single time steps.

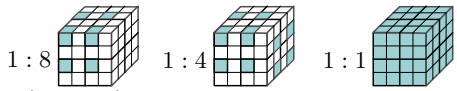


Figure 5: *8 $(2\times2\times2)$-cubes with different subsampling resolution. Blue: computed voxels, white: copied voxels.*

4. Implementation

Our TT4D tensor train decomposition is implemented in Python, using numpy and pytorch for matrix and tensor operations, and *teneva* for the alternating maximum volume approximation (maxvol) algorithm [ten, CRO23]. The real-time rendering pipeline is implemented in C++ using *VTK* for DVR [VTK] and parallelized tensor reconstruction performed directly on the GPU using Metal compute shaders.

The TT cores are stored as 16-bit floats and also reconstructed as such. The decomposition itself and the visualization are done with 32-bit floats, as we currently rely on external functionalities of pytorch and VTK that cannot handle 16-bit floats.

5. Result

We tested our method using the four datasets shown in Table 1. All experiments were conducted on a machine equipped with an Apple M2 chip (10-core) and 32GB of unified memory.

Name	Dimensions	Type	GB	Source
isotropic	128^4	float-32	1	[LPW*08, PBLM07]
jet	$129^2\times104\times150$	float-64	2	[GGYC11]
turbulence	$512^3\times64$	float-32	32	[LPW*08, PBLM07]
channel	$512^3\times128$	float-32	64	[LPW*08, PBLM07, GKY*16]

Table 1: *4D volume datasets used for testing.*

5.1. Decomposition

Table 2 shows the accuracy of our TT4D tensor compression approach for different compression ratios. For $\mathscr{T}$ being the input data and $\mathscr{T}'$ the reconstructed 4D data, the relative error is defined as $\text{norm}(\mathscr{T}-\mathscr{T}')/\text{norm}(\mathscr{T})$, the peak signal-to-noise ratio (PSNR) as $20\cdot\log_{10}((\max(\mathscr{T})-\min(\mathscr{T}))/\sqrt{\text{mean}((\mathscr{T}-\mathscr{T}')^2)})$, and the structural similarity index (SSIM) as proposed in [WBSS04]. The last rank of the compression, for the time dimension, can be defined lower than the first two ranks for the spatial dimensions to achieve faster reconstruction during rendering. Especially the high-resolution datasets *turbulence* and *channel* exhibit only moderate changes over time. However, that is typical for high-resolution scientific datasets and consequently allows for higher compression ratios with higher accuracy. But we note that this advantage holds for all compared methods. The compression ratios are given in comparison to the original data size.

As a baseline reference to state-of-the-art compression methods, we compare TT4D to three other methods: ZFP [Lin14], using the fixed-rate compression which allows for random partial

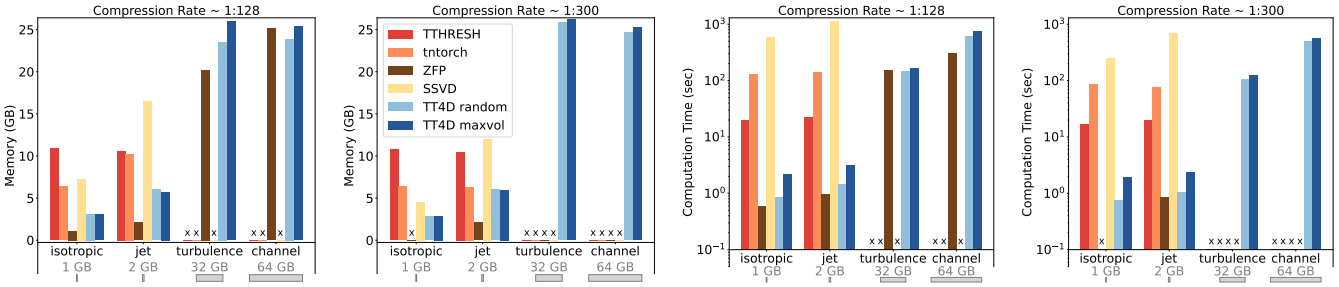


Figure 6: *Comparison of peak memory and runtime during compression of different methods with two exemplary compression ratios. "X" denotes values that we were not able to compute due to memory issues or too low provided compression ratios, as ZFP fixed-rate compresses only up to* $1:128$ *for 32-bit floats. Note that the computation time is given on a log scale.*

Dataset	compression ratio	ranks	relative error	PSNR (dB)	SSIM
isotropic	1:15	(88, 1894, 57)	0.2331	41.8430	0.9563
	1:297	(47, 195, 25)	0.3923	37.32	0.8595
jet	1:15	(90, 3613, 68)	0.2077	52.4951	0.9984
	1:294	(62, 305, 34)	0.4229	46.3173	0.9913
turbulence	1:50	(255, 2280, 18)	0.0281	61.2764	0.9995
	1:1000	(125, 250, 7)	0.1676	45.7819	0.981
channel	1:128	(384, 1258, 32)	0.2383	50.2252	0.9935
	1:1000	(172, 344, 21)	0.6416	41.6221	0.944

Table 2: *Accuracy after compression of different datasets with different compression ratios using TT4D maxvol column sampling.*

decompression, TTHRESH [BRLP20], and TT decomposition via tntorch [UBRS22]. We omit the comparison to the SZ compressor as ZFP has been shown to have very similar compression ratios and accuracy [BRLP20]. We also compare our maximum volume column sampling to plain random sampling and to a basic implementation of the sequential singular value decomposition (SSVD) without sampling.

Figure 6 compares the observed compression time and peak memory usage for two exemplary target compression ratios of $1:128$ and $1:300$. Note that ZFP fixed-rate compression does not allow for compression ratios more than $1:128$ for 32-bit floats. Moreover, due to too much memory consumption of some methods, we were not able to compress all datasets with all methods. All methods which were running out of memory or could not complete the task are indicated with $\times$ in the charts. As can be seen, our TT4D decomposition method has a significantly lower memory usage than TTHRESH, tntorch, and the default SSVD, leading to the possibility of processing notably larger datasets than these methods. Furthermore, our TT4D decomposition approaches are faster than all compared methods, except for ZFP at the lower compression rate. ZFP achieves a slightly faster compression using less memory but it can not compress more than $1:128$ for 32-bit float data, which is a disadvantage for large datasets. We can also see that random column sampling is slightly faster than the maximum volume approximation, due to the iterative nature of the latter. Random column sampling also uses slightly less memory than maxvol sampling due to additional auxiliary matrices that are needed in the implementation of the matrix maximum volume approximation [ten, CRO23] we used.

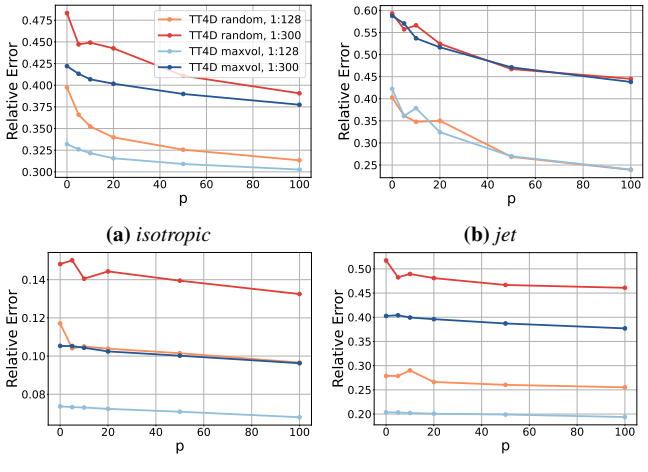
Putting the achievable compression ratios in relation to other methods, we observe that TTHRESH can achieve higher compression ratios, showing a similar accuracy with respect to the relative error (Table 3). This is to be expected since our method is not designed to achieve maximal compression, as we do not exploit any variable-bit length, sparse coding, or vector quantization. Instead, our method is optimized for low time and memory usage. Furthermore, our method allows more compression domain data processing operations than other methods. The experiments also show that we can achieve much better compression ratios for the same accuracy using no column subsampling. At the same time, the maximum volume approximation, compared to random column sampling, allows us to achieve almost twice the compression ratio. In our experiments, we used a maximum of 15 iterations and an oversampling parameter of 10 for the maximum volume approximation. Increasing these parameters could lead to compression ratios even closer to the basic SSVD. For the two larger datasets, *turbulence* and *channel*, TT4D maxvol sampling can provide the highest compression based on similar accuracy compared to the other methods that are capable of compressing such large datasets in these experiments.

Figure 7 shows the effect of our oversampling parameter p on the reconstruction accuracy. As expected, the relative error can be reduced with a higher oversampling parameter. However, increasing p requires the processing of larger matrices, which reduces the overall efficiency of the decomposition, and it has a limited effect on the larger data sets.

One key advantage of our decomposition is that after loading and partial initial decoding of the data, we do not need much additional memory, allowing us to decompose large datasets. Figure 8 shows the memory consumption over time, exemplary for the decomposition of the *isotropic* dataset. We compare random and maxvol column sampling, as well as tntorch and SSVD with no sampling. The red lines indicate the peak memory consumption for each method. Note that the time axes show the different overall decomposition running times, which are significantly longer for tntorch and SSVD, while TT4D random sampling is a bit faster than the maxvol variant. We can see that TT4D needs less memory in total as it does most operations in-place or on small submatrices, while tntorch and basic SSVD make copies of the data or rely on large intermediate matrices, and therefore use even more memory than the initial amount.

Dataset	relative error	ZFP fixed-rate	TTHRESH	tntorch	SSVD	TT4D random sampling	TT4D iterative maxvol sampling
isotropic	0.25	1 : 64	1 : 1,461	-	1 : 78	1 : 14	1 : 22
	0.50	1 : 128†	1 : 72,934	1 : 1,753	1 : 4,542	1 : 671	1 : 1,016
jet	0.20	*	1 : 2,279	-	1 : 47	1 : 21	1 : 44
	0.50	*	1 : 44,328	1 : 144	1 : 1,440	1 : 406	1 : 573
turbulence	0.02	*	-	-	-	1 : 12	1 : 23
	0.17	1 : 14	-	-	-	1 : 515	1 : 1,000
channel	0.25	1 : 16	-	-	-	1 : 89	1 : 150
	0.50	1 : 32	-	-	-	1 : 420	1 : 610

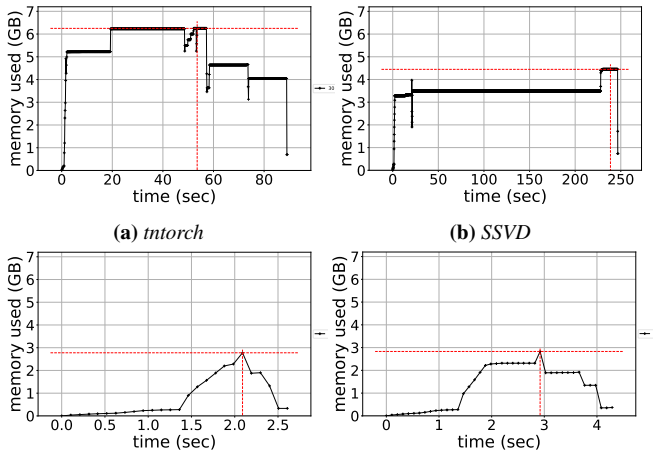
Table 3: *Comparison of compression ratios for the same relative reconstruction error. "-" denotes values the method was not able to complete due to memory or singularity issues. "*" denotes cases where the low target error could not be achieved. "†" indicates that* 1 : 128 *is the maximum compression ratio supported by ZFP fixed-rate compression for 32-bit floats and higher ratios are therefore unavailable.*



(a) *isotropic* **(b)** *jet*

(c) *turbulence* **(d)** *channel*

Figure 7: *Analysis of the oversampling parameter p for random and iterative maxvol column sampling.*



(a) *tntorch* **(b)** *SSVD*

(c) *TT4D random sampling* **(d)** *TT4D maxvol approx.*

Figure 8: *Memory profiling of decomposition of the* isotropic *dataset (1GB) over time with compression* 1 : 300*.*

dataset	ZFP fixed-rate	TTHRESH	naive reconstruction	TT4D per time step	TT4D (1 : 8)-subsampled reconstruction	TT4D partial decoding
isotropic	22.1	7,767	154	2.3	1.1	153
jet	18.8	8,547	334	2.3	1.1	334
turbulence	2,000	-	14,243	105	14.4	13,271
channel	1,997	-	36,332	156	20.8	34,675

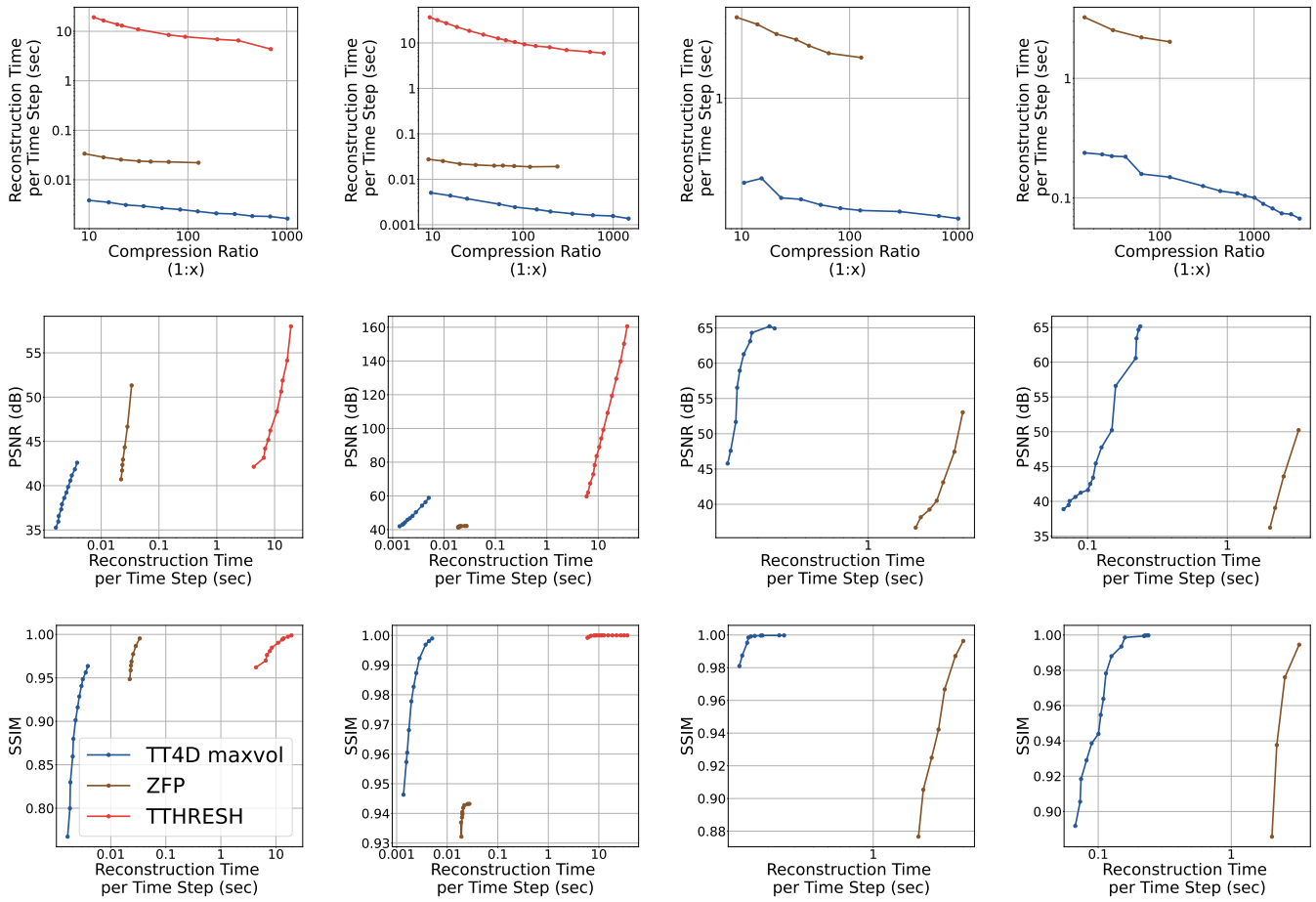
Table 4: *Reconstruction times in ms of different methods for a compression ratio of* 1 : 128*. For ZFP, we measured the time to decode the smallest possible amount of blocks, equalling* 4 *time steps, but report the time amortized for one time step. Naive reconstruction refers to reconstruction without any preprocessing, being the multiplication of the first 3 tensor cores with one column vector of the last tensor core. Subsampled reconstruction denotes the time to reconstruct one subsampled time step. Partial Decoding refers to the time needed to construct the base tensor as described in Section 3.2. Entries "-" denote cases where the compression was not possible due to memory issues.*

5.2. Reconstruction

We claim that the TT4D compression domain visualization approach is suitable for fast reconstruction, even of random single time steps, supporting interactive rendering. To demonstrate this, we show a comparison of reconstruction times for different compression techniques in Table 4. Since ZFP does not allow for single time step reconstruction, we measure the time for reconstructing 4 time steps, which is the smallest possible reconstruction size, but report the amortized time per time step. As we can see, our method provides fast full-resolution reconstruction of individual time steps and is much faster than the other compared methods. Even ZFP's amortized single time step reconstruction is still an order of magnitude slower than our TT4D. To demonstrate the effectiveness of our approach, we also report the naive reconstruction of individual time steps from the fully compressed TT format. In the last column, we additionally report the partial decoding time, which is a startup cost required only once when loading a new dataset. One can see that our partial decoding introduced in Section 3.2 reduces the reconstruction time per time step significantly.

Note that even with our fast reconstruction from a partial decoding, the reconstruction time for the much larger *turbulence* and the *channel* datasets can be more than 100ms. This demonstrates the necessity of our proposed subsampled reconstruction, see Section 3.2.1, to allow for fluent reconstruction during fast user interactions. For example, a 1 : 8 subsampled time slice of the (1 : 128)-compressed *channel* dataset can be decoded in an average time of only 20.8ms (see Table 4).

Furthermore, we state that TT4D allows for faster time step reconstruction under similar accuracy. Figure 9 shows that TT4D provides significantly faster reconstruction than ZFP fixed-rate and



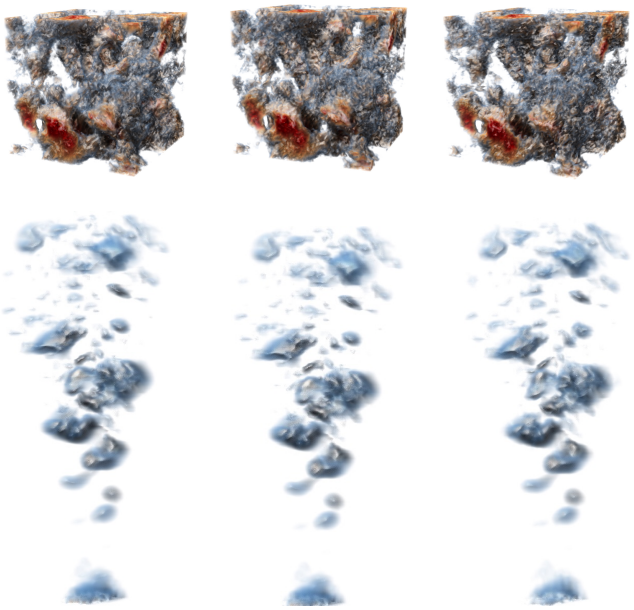
(a) *isotropic* **(b)** *jet* **(c)** *turbulence* **(d)** *channel*

Figure 9: *Comparison of compression ratio to reconstruction time and comparison of reconstruction time to accuracy in terms of PSNR and SSIM between ZFP, TTHRESH, and TT4D maxvol. For ZFP, we measure the time to decode the smallest possible amount of blocks, equalling* 4 *time steps, but report the time amortized for one time step. TTHRESH is not able to compress the* turbulence *and* channel *datasets. ZFP fixed-rate does not allow for higher compression than* $1:128$ *for 32-bit floating points. Note the logarithmic scale for the reconstruction times and compression ratios.*

TTHRESH for similar compression ratios. The graph also shows that for all but one dataset, we achieve higher accuracy in terms of PSNR and SSIM than ZFP. TTHRESH provides more accurate compression, however, the reconstruction times are too slow to support any interactive reconstruction and exploration of the dataset. Moreover, TTHRESH cannot handle the compression of large datasets, such as the *turbulence* and *channel* dataset.

In Figure 10, the reconstruction results of our method are compared visually to the original datasets and to the reconstruction results of TTHRESH. While our method can provide high-quality reconstructions of the key visual features, TTHRESH provides a slightly better reconstruction in the fine details and high frequencies. However, using our method, the 4D data is never completely reconstructed, leading to less memory usage and the possibility to decompose and visualize significantly larger datasets than with TTHRESH, as shown in Figure 11. It is important to note that, although we present our results using VTK for rendering, our method is independent of the rendering framework and could be integrated with different basic volume renderers of the user's choice.

Beyond basic reconstruction and rendering, the linearity of tensor decompositions allows for interactive and efficient application of convolutional filtering operations in decomposed space, avoiding processing the full-size dataset. It even allows for the application of filters along the time dimension without ever needing to reconstruct multiple time steps. For more information on the application of basic filtering operations to decomposed tensors, see [BRSP18]. Figure 12 shows the application of a one-dimensional Difference of Gaussian filter kernel along the time dimension, showing the areas of major changes over time. The application of the filter leads to virtually no additional effort, being computed in less than $2ms$ for the shown datasets.



(a) *original* **(b)** *TTHRESH* **(c)** *TT4D*

Figure 10: *Comparison of the reconstruction results of TT4D to TTHRESH and the original* isotropic *(top) and* jet *(bottom) data for a compression ratio of* 1 : 128.

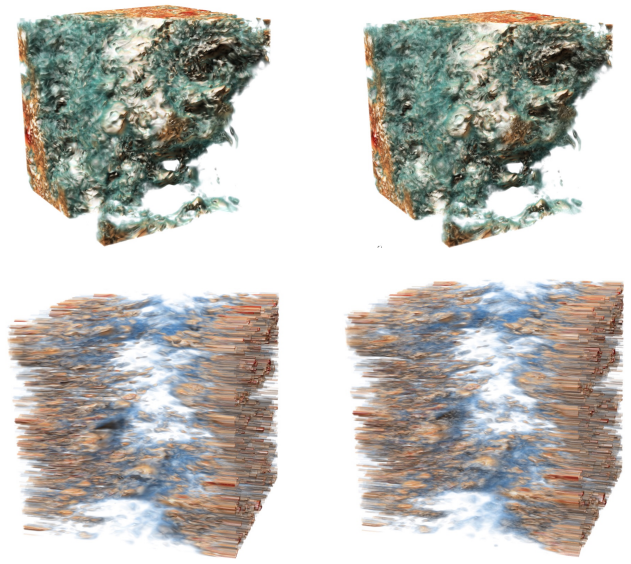
5.3. Limitations

While SSVD may achieve higher accuracy compared to TT4D's random or maxvol column sampling, it is impractical for larger 4D volume data. Nevertheless, more parameter tuning and higher oversampling parameters might help to close that difference. In any case, the accuracy of the results depends on the choice and distributions of the ranks, which are user-defined. Further exploration could help to find more optimal rank configurations that may lead to more accurate compression. A second limitation is that TT4D's smooth and flexible reconstruction relies on computing and storing an intermediate reduced-size fourth-order base tensor for fast rendering. This tensor is up to multiple factors smaller than the full 4D dataset, but for very large spatial dimensions and high decomposition ranks, it could still be large and a limiting factor.

6. Conclusion

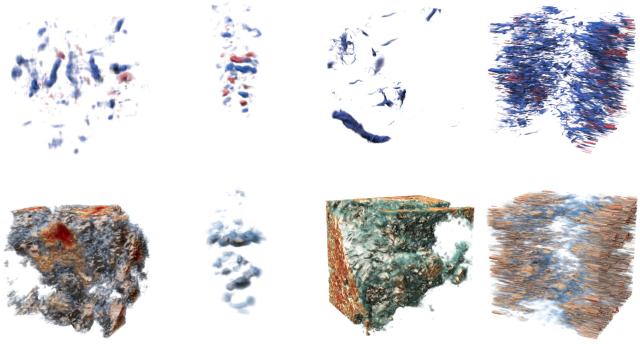
We introduced TT4D, a tensor train-based method for compressing 4D time-varying volume data. By leveraging different column subsampling strategies, TT4D operates on submatrices during the decomposition, avoiding expensive computations on large matrices and eliminating costly reshaping operations. Our results have shown that TT4D reduces memory and time consumption compared to existing state-of-the-art compression methods.

Moreover, TT4D introduces flexible, on-the-fly reconstruction of individual time steps without ever requiring the reconstruction of the full dataset. Through adaptive multiresolution reconstruction, TT4D enables real-time updates, outperforming naive reconstruction approaches, allowing for interactive exploration of the data even for large datasets. Exploiting the advantages of tensor trains,



(a) *original* **(b)** *TT4D*

Figure 11: *Comparison of reconstruction results to the original* turbulence *(top) and* channel *(bottom) data for a compression ratio of* 1 : 128.



(a) *isotropic* **(b)** *jet* **(c)** *turbulence* **(d)** *channel*

Time: 0.9*ms*, 0.7*ms*, 1.2*ms*, 1.4*ms*

Figure 12: *1D difference of Gaussians filter over time dimension. Reconstruction ratio* 1 : 128. *The indicated time refers to the time needed to apply the filter. Red regions indicate positive peaks, while blue regions indicate negative peaks. Top: filter, b ottom: reconstruction without filter.*

TT4D allows for efficient application of filter operations in the decomposed domain.

Acknowledgements

The authors would like to thank the Johns Hopkins University [JHT] as well as Sandia National Laboratories [GGYC11] for kindly providing the datasets we have used for testing.
Open access publishing facilitated by Universitat Zurich, as part of the Wiley - Universitat Zurich agreement via the Consortium Of Swiss Academic Libraries.

References

[ABK16] AUSTIN W., BALLARD G., KOLDA T. G.: Parallel tensor compression for large-scale scientific data. In *Proceedings IEEE International Parallel & Distributed Processing Symposium* (2016). doi:https://doi.org/10.1109/IPDPS.2016.67. 2

[BHP15] BEYER J., HADWIGER M., PFISTER H.: State-of-the-art in GPU-based large-scale volume visualization. *Computer Graphics Forum 34*, 8 (December 2015), 13–37. doi:10.1111/cgf.12605View. 2

[BKK20] BALLARD G., KLINVEX A., KOLDA T. G.: TuckerMPI: A parallel C++/MPI software package for large-scale data compression via the Tucker tensor decomposition. *ACM Transactions on Mathematical Software 46*, 2 (June 2020), 1–31. doi:https://doi.org/10.1145/3378445. 2

[BRGI*13] BALSA RODRÍGUEZ M., GOBBETTI E., IGLESIAS GUITIÁN J. A., MAKHINYA M., MARTON F., PAJAROLA R., SUTER S. K.: A survey of compressed GPU direct volume rendering. In *Eurographics State of The Art Reports (STAR)* (2013), pp. 117–136. URL: 10.2312/conf/EG2013/stars/117-136. 2

[BRGI*14] BALSA RODRÍGUEZ M., GOBBETTI E., IGLESIAS GUITIÁN J. A., MAKHINYA M., MARTON F., PAJAROLA R., SUTER S. K.: State-of-the-art in compressed GPU-based direct volume rendering. *Computer Graphics Forum 33*, 6 (2014), 77–100. URL: http://dx.doi.org/10.1111/cgf.12280, doi:10.1111/cgf.12280. 2

[BRLP20] BALLESTER-RIPOLL R., LINDSTROM P., PAJAROLA R.: TTHRESH: Tensor compression for multidimensional visual data. *IEEE Transactions on Visualization and Computer Graphics 26*, 9 (September 2020), 2891–2903. doi:https://doi.org/10.1109/TVCG.2019.2904063. 2, 7

[BRP16a] BALLESTER-RIPOLL R., PAJAROLA R.: Lossy volume compression using Tucker truncation and thresholding. *The Visual Computer 32*, 11 (November 2016), 1433–1446. doi:https://doi.org/10.1007/s00371-015-1130-y. 2

[BRP16b] BALLESTER-RIPOLL R., PAJAROLA R.: Structural volume inpainting via Tucker dictionary learning. In *Workshop on Tensor Decompositions and Applications* (January 2016). 3

[BRP16c] BALLESTER-RIPOLL R., PAJAROLA R.: Visual data processing in the tensor compressed domain. In *Workshop on Tensor Decompositions and Applications* (January 2016). 3

[BRP19] BALLESTER-RIPOLL R., PAJAROLA R.: Tensor decompositions for integral histogram compression and look-up. *IEEE Transactions on Visualization and Computer Graphics 25*, 2 (February 2019), 1435–1446. doi:https://doi.org/10.1109/TVCG.2018.2802521. 2, 3

[BRSP15] BALLESTER-RIPOLL R., SUTER S. K., PAJAROLA R.: Analysis of tensor approximation for compression-domain volume visualization. *Computers & Graphics 47* (April 2015), 34–47. doi:https://doi.org/10.1016/j.cag.2014.10.002. 2

[BRSP18] BALLESTER-RIPOLL R., STEINER D., PAJAROLA R.: Multiresolution volume filtering in the tensor compressed domain. *IEEE Transactions on Visualization and Computer Graphics 24*, 10 (October 2018), 2714–2727. doi:https://doi.org/10.1109/TVCG.2017.2771282. 2, 3, 9

[BTL20] BAI Z., TAO Y., LIN H.: Time-varying volume visualization: a survey. *Journal of Visualization 23*, 5 (June 2020), 745–761. doi:https://doi.org/10.1007/s12650-020-00654-x. 3

[CRO23] CHERTKOV A., RYZHAKOV G., OSELEDETS I.: Black box approximation in the tensor train format initialized by ANOVA decomposition. *SIAM Journal on Scientific Computing 45*, 4 (2023), A2101–A2118. doi:https://doi.org/10.1137/22M1514088. 6, 7

[CW19] CHE M., WEI Y.: Randomized algorithms for the approximations of Tucker and the tensor train decompositions. *Advances in Computational Mathematics 45*, 1 (2019), 395–428. doi:https://doi.org/10.1007/s10444-018-9622-8. 4

[DC16] DI S., CAPPELLO F.: Fast error-bounded lossy HPC data compression with SZ. In *Proceedings IEEE International Symposium on Parallel and Distributed Processing* (2016), pp. 730–739. doi:https://doi.org/10.1109/IPDPS.2016.11. 2

[FM07] FOUT N., MA K.-L.: Transform coding for hardware-accelerated volume rendering. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1600–1607. doi:10.1109/TVCG.2007.70606. 2

[GEA96] GROSSO R., ERTL T., ASCHOFF J.: Efficient data structures for volume rendering of wavelet-compressed data. In *Proceedings Winter School of Computer Graphics* (1996), Computer Society Press. 2

[GG16] GUTHE S., GOESELE M.: Variable length coding for GPU-based direct volume rendering. In *Proceedings Vision, Modeling and Visualization* (2016), pp. 77–84. doi:https://doi.org/10.2312/vmv.20161345. 2

[GGYC11] GROUT R., GRUBER A., YOO C. S., CHEN J.: Direct numerical simulation of flame stabilization downstream of a transverse fuel jet in cross-flow. *Proceedings of the Combustion Institute 33*, 1 (December 2011), 1629–1637. doi:https://doi.org/10.1016/j.proci.2010.06.013. 6, 10

[GKY*16] GRAHAM J., KANOV K., YANG X. I. A., LEE M., MALAYA N., LALESCU C. C., BURNS R., EYINK G., SZALAY A., MOSER R. D., MENEVEAU C.: A web services accessible database of turbulent channel flow and its use for testing a new integral wall model for LES. *Journal of Turbulence 17*, 2 (2016), 181–215. doi:https://doi.org/10.1080/14685248.2015.1088656. 6

[GLDH97] GROSS M. H., LIPPERT L., DITTRICH R., HÄRING S.: Two methods for wavelet-based volume rendering. *Computers & Graphics 21*, 2 (March/April 1997), 237–252. doi:https://doi.org/10.1016/S0097-8493(96)00087-8. 2

[HAAB*18] HADWIGER M., AL-AWAMI A., BEYER J., AGUS M., PFISTER H.: Sparseleap: Efficient empty space skipping for large-scale volume rendering. *IEEE Transactions on Visualization and Computer Graphics 24*, 1 (January 2018), 974–983. doi:10.1109/TVCG.2017.2744238. 2

[IAVHDL11] ISHTEVA M., ABSIL P.-A., VAN HUFFEL S., DE LATHAUWER L.: Tucker compression and local optima. *Chemometrics and Intelligent Laboratory Systems 106*, 1 (March 2011), 57–64. doi:https://doi.org/10.1016/j.chemolab.2010.06.006. 2

[IP99] IHM I., PARK S.: Wavelet-based 3D compression scheme for interactive visualization of very large volume data. *Computer Graphics Forum 18*, 1 (1999), 3–15. doi:https://doi.org/10.1111/1467-8659.00298. 2

[JHT] Johns Hopkins turbulence database (JHTDB). https://turbulence.idies.jhu.edu/home. 10

[KS99] KIM T.-Y., SHIN Y.-G.: An efficient wavelet-based compression method for volume rendering. In *Proceedings Pacific Graphics* (1999), pp. 147–156. doi:https://doi.org/10.1109/PCCGA.1999.803358. 2

[Lin14] LINDSTROM P.: Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (December 2014), 2674–2683. doi:https://doi.org/10.1109/TVCG.2014.2346458. 2, 6

[LJLB21] LU Y., JIANG K., LEVINE J. A., BERGER M.: Compressive neural representations of volumetric scalar fields. *Computer Graphics Forum 40*, 3 (2021), 135–146. doi:https://diglib.eg.org/handle/10.1111/cgf14295. 3

[LK11] LAINE S., KARRAS T.: Efficient sparse voxel octrees. *IEEE Transactions on Visualization and Computer Graphics 17*, 8 (August 2011), 1048–1059. doi:https://doi.org/10.1109/TVCG.2010.240. 2

[LLLZ22] LIU Y., LIU J., LONG Z., ZHU C.: *Tensor Computation for Data Analysis*. Springer, 2022. 3, 5

[LMC01] LUM E., MA K.-L., CLYNE J.: Texture hardware assisted rendering of time-varying volume data. In *Proceedings IEEE Visualization* (2001), pp. 263–270. doi:https://doi.org/10.1109/VISUAL.2001.964520. 3

[LPW*08] LI Y., PERLMAN E., WAN M., YANG Y., MENEVEAU C., BURNS R., CHEN S., SZALAY A., EYINK G.: A public turbulence database cluster and applications to study lagrangian evolution of velocity increments in turbulence. *Journal of Turbulence 9*, 31 (April 2008). doi:https://doi.org/10.1080/14685240802376389. 6

[MAG19] MARTON F., AGUS M., GOBBETTI E.: A framework for GPU-accelerated exploration of massive time-varying rectilinear scalar volumes. *Computer Graphics Forum 38*, 3 (June 2019), 53–66. doi:https://doi.org/10.1111/cgf.13671. 3

[MO15] MIKHALEV A. Y., OSELEDETS I. V.: Rectangular maximum-volume submatrices and their applications. *Linear Algebra and its Applications 538*, Supplement C (2015), 187–211. doi:10.1016/j.laa.2017.10.014. 5

[Mur93] MURAKI S.: Volume data and wavelet transform. *IEEE Computer Graphics and Applications 13*, 4 (July 1993), 50–56. doi:https://doi.org/10.1109/38.219451. 2

[NH92] NING P., HESSELINK L.: Vector quantization for volume rendering. In *Proceedings ACM Workshop on Volume Visualization* (1992), pp. 69–74. doi:https://doi.org/10.1145/147130.147152. 2

[Ose11] OSELEDETS I. V.: Tensor-train decomposition. *SIAM Journal on Scientific Computing 33*, 5 (September 2011), 2295–2317. doi:https://doi.org/10.1137/090752286. 2, 3, 5

[OT10] OSELEDETS I. V., TYRTYSHNIKOV E.: TT-cross approximation for multidimensional arrays. *Linear Algebra Applications 432*, 1 (2010), 70–88. doi:https://doi.org/10.1016/j.laa.2009.07.024. 4

[PBLM07] PERLMAN E. A., BURNS R. C., LI Y., MENEVEAU C.: Data exploration of turbulence simulations using a database cluster. In *Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing* (November 2007), ACM Press, p. 23. doi:https://doi.org/10.1145/1362622.1362654. 6

[RK09] RUITERS R., KLEIN R.: BTF compression via sparse tensor decomposition. *Computer Graphics Forum 28*, 4 (June 2009), 1181–1188. doi:https://doi.org/10.1111/j.1467-8659.2009.01495.x. 2

[RPD22] RAPP T., PETERS C., DACHSBACHER C.: Image-based visualization of large volumetric data using moments. *IEEE Transactions on Visualization and Computer Graphics 28*, 6 (June 2022), 2314–2325. doi:https://doi.org/10.1109/TVCG.2022.3165346. 3

[SKMH14] SICAT R., KRÜGER J., MÖLLER T., HADWIGER M.: Sparse PDF volumes for consistent multi-resolution volume rendering. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (December 2014), 2417–2426. doi:https://doi.org/10.1109/TVCG.2014.2346324. 2

[SMP13] SUTER S. K., MAKHINYA M., PAJAROLA R.: TAMRESH: Tensor approximation multiresolution hierarchy for interactive volume visualization. *Computer Graphics Forum 32*, 3 (2013), 151–160. doi:http://dx.doi.org/10.1111/cgf.12102. 3

[ten] teneva. https://teneva.readthedocs.io. 6, 7

[UBRS22] USVYATSOV M., BALLESTER-RIPOLL R., SCHINDLER K.: tntorch: Tensor network learning with PyTorch. *Journal of Machine Learning Research 23*, 208 (https://github.com/rballester/tntorch 2022), 1–6. URL: https://github.com/rballester/tntorch, doi:https://doi.org/10.48550/arXiv.2206.11128. 7

[VTK] Visualization toolkit (VTK). https://vtk.org. 6

[WBSS04] WANG Z., BOVIK A. C., SHEIKH H. R., SIMONCELLI E. P.: Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (April 2004), 600–612. doi:https://doi.org/10.1109/TIP.2003.819861. 6

[WHW22] WEISS S., HERMÜLLER P., WESTERMANN R.: Fast neural representations for direct volume rendering. *Computer Graphics Forum 41*, 6 (September 2022), 196–211. doi:https://doi.org/10.1111/cgf.14578. 3

[WWS*05] WANG H., WU Q., SHI L., YU Y., AHUJA N.: Out-of-core tensor approximation of multi-dimensional matrices of visual data. *ACM Transactions on Graphics 24*, 3 (July 2005), 527–535. doi:https://doi.org/10.1145/1073204.1073224. 2

[WXC*08] WU Q., XIA T., CHEN C., LIN H.-Y. S., WANG H., YU Y.: Hierarchical tensor approximation of multidimensional visual data. *IEEE Transactions on Visualization and Computer Graphics 14*, 1 (January/February 2008), 186–199. doi:https://doi.org/10.1109/TVCG.2007.70406. 2

[WYM10] WANG C., YU H., MA K.-L.: Application-driven compression for visualizing large-scale time-varying data. *IEEE Computer Graphics and Applications 30*, 1 (January/February 2010), 59–69. doi:https://doi.org/10.1109/MCG.2010.3. 3

[YL95] YEO B.-L., LIU B.: Volume rendering of DCT-based compressed 3D scalar data. *IEEE Transactions on Visualization and Computer Graphics 1*, 1 (1995), 29–43. doi:https://doi.org/10.1109/2945.468390. 2

[YZW*17] YU S., ZHANG S., WANG K., XIA Y., ZHANG H.: An efficient and fast GPU-based algorithm for visualizing large volume of 4D data from virtual heart simulations. *Biomedical Signal Processing and Control 35* (May 2017), 8–18. doi:https://doi.org/10.1016/j.bspc.2017.01.015. 3