LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-JRNL-750580

TTHRESH: Tensor Compression for Multidimensional Visual Data

R. Ballester-Ripoll, P. Lindstrom, R. Pajarola

May 1, 2018

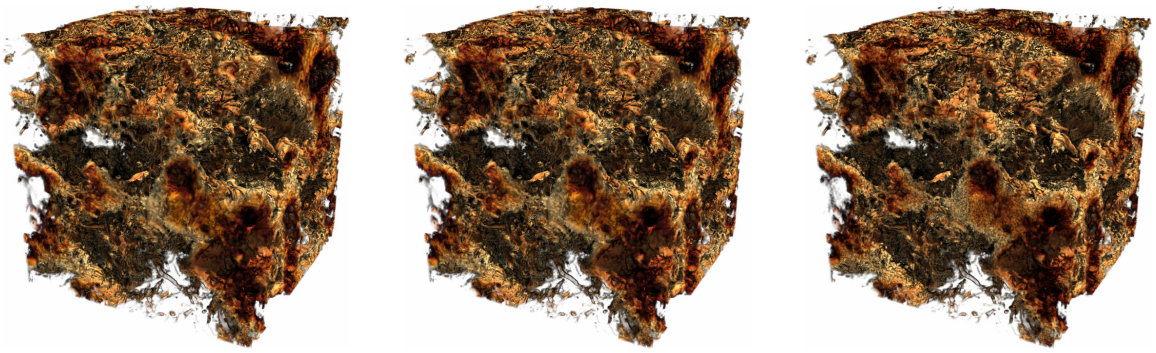
IEEE Transactions on Visualization and Computer Graphics

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

TTHRESH: Tensor Compression for Multidimensional Visual Data

Rafael Ballester-Ripoll, *Member, IEEE,* Peter Lindstrom, *Senior Member, IEEE*,
and Renato Pajarola, *Senior Member, IEEE*



(a) Original (512MB) (b) 10:1 compression (51.2MB) (c) 300:1 compression (1.71MB)

Fig. 1. (a) a 512^3 isotropic turbulence volume [1]; (b) visually identical compression result; (c) result after extreme compression.

Abstract—Memory and network bandwidth are decisive bottlenecks when handling high-resolution multidimensional data sets in visualization applications, and they increasingly demand suitable data compression strategies. We introduce a novel lossy compression algorithm for multidimensional data over regular grids. It leverages the higher-order singular value decomposition (HOSVD), a generalization of the SVD to three dimensions and higher, together with bit-plane, run-length and arithmetic coding to compress the HOSVD transform coefficients. Our scheme degrades the data particularly smoothly and achieves lower mean squared error than other state-of-the-art algorithms at low-to-medium bit rates, as it is required in data archiving and management for visualization purposes. Further advantages of the proposed algorithm include very fine bit rate selection granularity and the ability to manipulate data at very small cost in the compression domain, for example to reconstruct filtered and/or subsampled versions of all (or selected parts) of the data set.

Index Terms—Transform-based compression, scientific visualization, higher-order singular value decomposition, Tucker model, tensor decompositions

1 Introduction

Most scientific and visual computing applications face heavy computational and data management challenges when handling large and/or complex data sets over Cartesian grids. Limitations in memory resources or available transmission throughput make it crucial to reduce and compress such data sets in an efficient manner. Lossy compression is often the prescribed strategy, since many applications admit a certain error (especially for higher bit depths and floating-point precision). If the compressed data set is to be used for subsequent computational analysis and/or to be fed as the initial state of a simulation routine, only small errors are typically tolerated. Conversely, if visualization and user exploration are to follow decompression, then higher error rates are acceptable; the method developed in this paper is mainly geared towards this case. Depending on the specific application, certain additional properties are sometimes desired. These may include fast support for random-access decompression, fine compression rate granularity, asymmetry (faster decompression than compression), bounded error, support for arbitrary dimensionality, ease of parallelization, topological robustness, etc. These aspects make multidimensional compression a broad and challenging problem for which, unsurprisingly, no catch-all solution exists.

In this context, *tensor decompositions* and in particular the *Tucker model* are promising mathematical tools for higher-order compression and dimensionality reduction in the fields of graphics and visualization. 3D scalar field compression at the Tucker transform coefficients level was recently investigated [2], and it was concluded that coefficient thresholding outperforms earlier rank truncation-based approaches in terms of quality vs. compression ratio. This has motivated us to develop and introduce TTHRESH, a novel lossy compressor based on the Tucker decomposition. It is the first of its kind that supports arbitrary target accuracy via bit-plane coding. Previous related approaches fixed a number of quantization bits per transform coefficient, and sometimes even the transform basis size (the *tensor ranks*). Instead, our method drastically improves the compression ratio-accuracy trade-off curve by greedily compressing bit planes of progressively less importance. We also extend our encoding scheme to compress the factor matrices. The importance of this is unique to the HOSVD transform, which needs to store its learned bases as opposed to fixed-basis methods, yet never optimized by earlier works.

We also benchmark TTHRESH against other state-of-the-art compressors that are not based on the HOSVD. While the ratios we achieve at low error tolerances are comparable to those, we significantly outperform them on the higher error ranges on which visualization tasks usually rely.

We have released an open-source C++ implementation of our algo-

- *R. Ballester-Ripoll and R. Pajarola are with the Department of Informatics, University of Zürich, Switzerland. E-mails: rballester@ifi.uzh.ch and pajarola@ifi.uzh.ch.*
- *Peter Lindstrom is with the Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, USA. E-mail: pl@llnl.gov.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

rithm[1]. It is primarily intended as a standalone command-line utility, although its main functions are also usable in a header-only library fashion.

2 Related Work

2.1 3D Compression Algorithms

A number of lossy compression algorithms for scientific volume data sets have been proposed in the recent literature. For instance, ISABELA [3] focuses on spatio-temporal data with ample high-frequency components; it proceeds by sorting elements into a monotonic curve which is then fitted using B-splines. A more recent example of linearization strategy is SZ [4], which either predicts each coefficient using low-degree polynomials on preceding coefficients, or truncates it in its IEEE 754 binary representation. Some methods prioritize preserving specific properties of the data set, for example bounded error using topological features [5] or over connected and coherent regions (e.g. SQ [6]). Vector quantization [7, 8] requires heuristics or greedy algorithms during compression, but is fast to decompress and thus suitable for compression-domain direct volume rendering; see also the survey [9]. In particular, [7] was defined within an octree multiresolution hierarchy for fast ray-casting in an interactive volume visualization application.

A popular and long-standing family of compression methods are the ones that exploit linear transforms, including well-known decompositions such as the Fourier and discrete cosine transforms [10] and, since the 1990s, wavelets [11–15]. They capitalize on carefully designed transform bases that aim to sparsify real-world signals as much as possible. VAPOR [16], for example, uses a flexible wavelet-based compression layer integrated into an interactive volume and flow exploration tool. ZFP [17] is a floating-point compressor that uses custom transform matrices and emphasizes fast random access and low error for, among other applications, storing snapshots and intermediate steps in a computational analysis/processing pipeline. ZFP offers a transparent interface for compressed C/C++ arrays and operates via fixed-rate encoding, although a variable-rate variant is also supported.

2.2 Compressors Based on Tensor Decomposition

Several transform-based compression algorithms have been recently proposed that use data-dependent bases (the so-called *factor matrices*) instead of predefined ones. This is precisely the idea behind principal component analysis (PCA) as well as the Tucker decomposition. The Tucker model seeks to improve transform-domain sparsity at the expense of having to store its learned bases, which tends to be comparatively small for a three or more dimensions. Some of the earliest Tucker-based compression approaches for visual data include [18], [19] and [20]. Progressive tensor rank reduction (the so-called *truncation*; see later sections) has been shown to reveal features and structural details at different scales in volume data [21]. Further recent efforts in the context of tensor compression include [2, 9, 22–24] for interactive volume rendering and visualization, [25] for 3D displays, [26] for integral histograms of images and volumes, and [27–30] for reflectance fields, among others. The large-scale renderer TAMRESH [23] resembles block-transform coding in that the input volume is partitioned in small multiresolution cubic bricks; each brick is then compressed as a separate HOSVD core. Recently, Tucker core hard thresholding combined with factor matrix quantization was shown [2] to yield better compression rate than slice-wise truncating the core. These points have motivated the compressor proposed here.

3 Tucker/HOSVD Decomposition

Throughout this paper, tensors refer to multiarrays of dimension $N \geq 1$. We write vectors (tensors of dimension 1) in bold lowercase as in $\boldsymbol{x} = (x_1, \ldots, x_N)$, matrices (tensors of dimension 2) in bold capitals such as $\mathbf{U}$, and general tensors as well as sets in calligraphic letters such as $\mathcal{T}$. We generally use the notation and definitions from [31]; in particular, rows and columns in matrices generalize to tensors as *fibers*. The *n-th mode unfolding* of a tensor $\mathcal{T}$ arranges all n-mode fibers next to each other as columns of a wide matrix and is denoted as $\mathbf{T}_{(n)}$. The *tensor-times-matrix product* (TTM) contracts a tensor's n-mode fibers along a matrix's rows and is denoted as $\mathcal{T} \times_n \mathbf{U}$. We access tensors using bracket notation, so for instance $\mathbf{U}[1, 1]$ is the top left element of a matrix $\mathbf{U}$. We refer the reader to Kolda and Bader's survey [32] for more extensive details on basic tensor manipulation.

3.1 The Tucker Model

The *full* Tucker decomposition [31,33] writes any entry $\mathcal{T}[x_1, \ldots, x_N]$ of a 3D tensor $\mathcal{T}$ exactly as:

$$\sum_{r_1,\ldots,r_N=1}^{I_1,\ldots,I_N} \mathcal{B}[r_1, \ldots, r_N] \cdot \mathbf{U}^{(1)}[x_1, r_1] \cdots \mathbf{U}^{(N)}[x_N, r_N] \qquad (1)$$

or, in the more compact TTM notation,

$$\mathcal{T} = \mathcal{B} \times_1 \mathbf{U}^{(1)} \times_2 \cdots \times_N \mathbf{U}^{(N)} \qquad (2)$$

where each $\mathbf{U}^{(n)}$ is a non-singular matrix of size $I_n \times I_n$ and $\mathcal{B}$ is a *core* tensor of coefficients with the same size as $\mathcal{T}$. See Fig. 2(b) for an illustration of the full Tucker decomposition. The matrices $\mathbf{U}^{(n)}$ are called *Tucker factors* (or *factor matrices*) and define a two-way transformation between $\mathcal{T}$ and its core $\mathcal{B}$, whereby Eq. 2 is inverted as

$$\mathcal{B} = \mathcal{T} \times_1 \mathbf{U}^{(1)^{-1}} \times_2 \cdots \times_N \mathbf{U}^{(N)^{-1}}. \qquad (3)$$

The higher-order singular value decomposition (HOSVD) [31, 32] is an efficient procedure to construct orthogonal Tucker factors (i.e., whose columns are orthogonal unit vectors) by setting each $\mathbf{U}^{(n)}$ as the left singular vectors of the n-th mode unfolding matrix $\mathbf{T}_{(n)}$. In other words, the HOSVD sets each n-th factor as the uncentered PCA transformation matrix of the set of all fibers from $\mathcal{T}$, taken along the n-th mode. The Tucker model is flexible and readily applicable to any shape and dimensionality, and the HOSVD decomposition always exists.

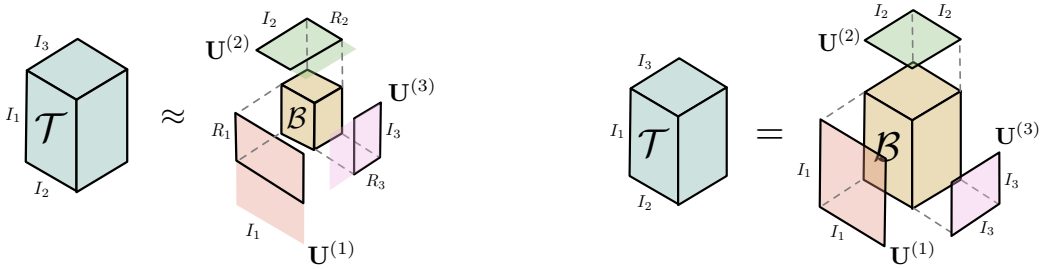
Since for three and more dimensions the core holds far more coefficients than the factors, it is also the decomposition part where most data reduction can be achieved and, consequently, the main source of error. Fortunately, we can determine and bound the l^2 error (i.e. sum of squared errors, or SSE for short) that is due to the core by just looking at its coefficients. Factor orthogonality implies that $\|\mathcal{T}\| = \|\mathcal{B}\|$. Furthermore, any perturbation in the core propagates directly to the reconstruction: $\|\widetilde{\mathcal{B}} - \mathcal{B}\| = \|\widetilde{\mathcal{T}} - \mathcal{T}\|$; see e.g. [31, 32]. This property will be crucial for our compression strategy.

3.2 Sparsifying Properties

Tucker-based compression algorithms exploit the fact that the HOSVD transform coefficients generated in $\mathcal{B}$ tend to be quasi-sparse for typical real-world or simulated multidimensional signals. In addition, many transformations do not significantly affect the HOSVD. For example, if one permutes some slices of $\mathcal{T}$ along one or more dimensions, its HOSVD will produce the same core $\mathcal{B}$ and factors (with their corresponding rows permuted). Other possible transformations that can be encoded on a HOSVD-compressed data set without essentially affecting $\mathcal{B}$ include spatially moving or stretching the data, padding it with zeros, upsampling it with multilinear interpolation, scaling by a constant (this will scale $\mathcal{B}$), etc. Many usual data reduction approaches are guaranteed to actually improve HOSVD core sparsity, including downsampling, box-filtered decimation, convolving with any band-limited kernel, etc. For instance, a volume whose k last wavelet levels are zero can be represented using a Tucker core with 8^k times fewer non-zero coefficients than otherwise needed.

The HOSVD decomposition decorrelates the data at all spatial scales, but does so without explicit space partitioning, i.e. avoiding tree-like structures or predefined multiresolution filter banks. The task of capturing correlation at multiple scales is thus undertaken by the different factor matrix columns. Nonetheless, the question of how to organize the coefficients in $\mathcal{B}$ for effective compression is unclear *a priori*.

[1] Available (LGPL-3.0) at `https://github.com/rballester/tthresh`.



(a) Core truncation

(b) Full core, used in this paper for subsequent processing and quantization

Fig. 2. Left: the Tucker rank truncation approach for 3D compression used in e.g. [20], [22], [23] and [24]. Right: the full core approach first considered in [2] and here extended into a full-fledged compressor with adaptive thresholding and bit-plane coding.

3.3 Core Truncation and Its Limitations

Conveniently, the HOSVD produces core (hyper-)slices that are non-increasing in norm. Let us consider the norm of each k-th slice of the Tucker core along the n-th dimension:

$$\sigma_k^{(n)} := \|\mathcal{T}[\overbrace{:,\ldots,:}^{n-1}, k, \overbrace{:,\ldots,:}^{N-n}]\|. \tag{4}$$

These norms have been proposed as *generalized singular values*, and they satisfy [31]:

$$\sigma_1^{(n)} \geq \sigma_2^{(n)} \geq \ldots \geq \sigma_{I_n}^{(n)} \geq 0. \tag{5}$$

Furthermore, from the factor matrix orthogonality it follows that the mean squared error (MSE) induced by zeroing-out a core coefficient is proportional to its squared magnitude. These properties have been exploited in the past as the basis of several truncation-based HOSVD compression schemes [20], [22], [23], [24], whereby the least important trailing factor columns in each $\mathbf{U}^{(n)}$ and the corresponding core slices in $\mathcal{B}$ along each dimension are discarded to produce a compressed approximation. By doing so, only $1 \leq R_n < I_n$ factor columns and core slices remain for each mode n (see Fig. 2(a)). The quantities $R_1, \ldots, R_N$ are known as *truncated Tucker ranks* and were used in those works for variable-detail compression and progressive reconstruction.

There are, however, two notable aspects that have not been pursued satisfactorily by these previous approaches. First, although the slice truncation idea is sound as motivated by Eq. 5, its granularity is very coarse. Elimination strategies on a coefficient-by-coefficient basis (rather than slice-by-slice) have the potential to significantly improve compression quality. Second, and regardless of the coefficient elimination method chosen, how to encode the surviving coefficients remains an open issue as well. Based on the roughly exponential growth of those coefficients, Suter et al. [22] proposed a fixed-bit logarithmic quantization scheme: 1 bit for the coefficient sign and 8 or 16 for the logarithm of its absolute value. The authors realized the extreme importance of the first element $\mathcal{B}[1,1,1]$ (the so-called *hot corner*, as shown in Fig. 3); it often captures most of the signal's energy and $\|\mathcal{B}[1,1,1]\| \approx \|\mathcal{T}\|$. Hence this value was saved separately at 64-bit floating-point precision. This strategy was later replicated in other works [2, 23]. Nevertheless, a truly adaptive compression approach for the full-length HOSVD core has not been explored as of yet. The strategy we propose builds on the thresholding-oriented analysis of [2] in that we compress coefficients, one bit plane p at a time, up to a certain plane $63 \geq P \geq 0$. In particular, elements whose absolute value is below 2^P are thresholded away. We give the full details in the following section; see also Fig. 2(b).

4 Proposed Algorithm

Let $\mathcal{T}$ be an input tensor and $\widetilde{\mathcal{T}}$ the result after compression and decompression. Our pipeline accepts one main compression parameter, namely the error target, which can be specified in one of three ways:

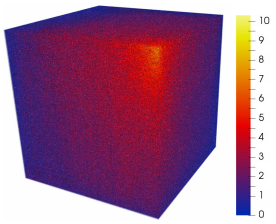


Fig. 3. HOSVD core $\mathcal{B}$ of size 256^3, obtained from the Foot data set. For visualization we scale all values $x \mapsto \ln(1+x)$, then apply the colormap shown on the right. Note the *hot corner* phenomenon.

- Relative error (`-e` flag; sometimes known as *normalized root mean square error*):

  $$\epsilon(\mathcal{T}, \widetilde{\mathcal{T}}) := \|\mathcal{T} - \widetilde{\mathcal{T}}\| / \|\mathcal{T}\|,$$

  where $\|\cdot\|$ denotes the Frobenius norm (i.e. the Euclidean norm of the flattened tensor).

- Root-mean-square error (`-r` flag):

  $$\mathrm{RMSE}(\mathcal{T}, \widetilde{\mathcal{T}}) := \|\mathcal{T} - \widetilde{\mathcal{T}}\| / \sqrt{I_1 \cdots I_N}.$$

- Peak signal-to-noise ratio (`-p` flag):

  $$\mathrm{PSNR}(\mathcal{T}, \widetilde{\mathcal{T}}) := 20 \cdot \log_{10} \left(\frac{\max\{\mathcal{T}\} - \min\{\mathcal{T}\}}{2 \cdot \mathrm{RMSE}(\mathcal{T}, \widetilde{\mathcal{T}})} \right).$$

The target specified is then converted to sum of squared errors (SSE) for the algorithm's internal use via the following equivalences:

$$\mathrm{SSE} = \epsilon^2 \cdot \|\mathcal{T}\|^2 = \mathrm{RMSE}^2 \cdot C = \left(\frac{\max\{\mathcal{T}\} - \min\{\mathcal{T}\}}{2 \cdot 10^{\mathrm{PSNR}/20}} \right)^2 \cdot C \tag{6}$$

where C is the total number of grid points $I_1 \cdots I_N$.

The algorithm consists of three main stages. First, the full non-truncated HOSVD is run on the input data set to yield N orthogonal square factor matrices and an N-dimensional core of the same size as the original. The HOSVD core is flattened as a 1D vector of C coefficients, which are then scaled and cast as 64-bit integers. We use C-ordering, i.e. dimensions in the core are traversed from right to left. Conceptually, we handle that sequence of integers as a $C \times 64$ binary matrix $\mathbf{M}$. Second, a number of that matrix's leftmost columns (the *bit planes*) are compressed without loss, namely the least number such that the overall l^2 error falls under a given target. This compression is achieved via run-length encoding (RLE) followed by arithmetic coding (AC). Last, the factor matrices are compressed using a cost-efficient budget criterion. See Algs. 1 and 2 for a pseudocode of our compression pipeline; its individual building blocks are detailed next.

Algorithm 1 Compress an N-dimensional tensor $\mathcal{T}$ of size $I_1 \times \cdots \times I_N$ at a prescribed sum of squared errors s using TTHRESH.

$\mathcal{B} := \mathcal{T}$
// HOSVD transform
for $n = 1, \ldots, N$ **do**
  $\mathbf{B}_{(n)} := \text{unfold}(\mathcal{B}, n)$ *// Size* $I_n \times (I_1 \cdots \widehat{I_n} \cdots I_N)$
  $\widehat{\mathbf{B}}_{(n)} := \mathbf{B}_{(n)} \cdot \mathbf{B}_{(n)}^T$ *// Symmetric matrix of size* $I_n \times I_n$
  $\mathbf{\Lambda}^{(n)}, \mathbf{U}^{(n)} = \text{eig}(\widehat{\mathbf{B}}_{(n)})$ *// Full decomposition; eigenvalues* $\mathbf{\Lambda}^{(n)}$ *in non-increasing order*
  $\mathbf{B}_{(n)} := {\mathbf{U}^{(n)}}^T \cdot \mathbf{B}_{(n)}$ *// Right part* $\mathbf{\Sigma} \cdot \mathbf{V}^T$ *of the SVD*
  $\mathcal{B} := \text{fold}(\mathbf{B}_{(n)})$ *// Back to original size*
end for
// $\mathcal{B}$ *is now the HOSVD core, and* $\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}$ *its factors*
$\alpha_b := \text{ENCODE}(\mathcal{B}, s)$ *// See Alg. 2*
for $n = 1, \ldots, N$ **do**
  $\text{ENCODE}(\mathbf{U}^{(n)}, \alpha_b)$
end for

Algorithm 2 Encode the decomposition parts obtained in Alg. 1. The input x can be either the core $\mathcal{B}$ or a factor $\mathbf{U}^{(n)}$.

function $\text{ENCODE}(x, \alpha_b, s)$
  $\mathcal{M} := \emptyset$ *// Mask to record coefficients that have already become significant. It starts out empty*
  $\mathbf{M} :=$ binary matrix of size $C \times 64$ containing all elements from x, in 64-bit unsigned integer format
  $\tilde{s} := \|x\|^2$ *// We start with the largest SSE*
  // Bit planes from more to less significant
  for $p = 63, \ldots, 0$ **do**
    for $c = 1, \ldots, C$ **do**
      if $c \in \mathcal{M}$ **then** *// The c-th coefficient is already significant*
        $\text{encodeBitVerbatim}(\mathbf{M}[c, p+1])$
      else
        $\text{encodeBitRLE}(\mathbf{M}[c, p+1])$
        if $\mathbf{M}[c, p+1] == 1$ **then** *// It becomes significant now*
          $\mathcal{M} := \mathcal{M} \cup \{c\}$
        end if
      end if
      Update current SSE $\tilde{s}$
      Estimate current ratio $\tilde{\alpha}$: the reduction in SSE achieved by the last c bits, divided by the number of bits needed to compress them
      if $\text{isCore}(x)$ and $\tilde{s} \leq s$ **then**
        Exit the two nested loops
      end if
      if $\text{isFactor}(x)$ and $\tilde{\alpha} \leq \alpha_b$ **then**
        Exit the two nested loops
      end if
    end for
  end for
  if $\text{isCore}(x)$ **then**
    return $\tilde{\alpha}$
  end if
end function

4.1 HOSVD Transform

We use the HOSVD as presented in Sec. 3 to compute orthogonal Tucker factors as the left singular vectors of unfolding matrices. We use 64-bit floating point precision. In general one may directly compute these singular vectors from each unfolding $\mathbf{B}_{(n)} = \mathbf{U}^{(n)} \cdot \mathbf{\Sigma}^{(n)} \cdot {\mathbf{V}^{(n)}}^T$ in one run of any standard SVD algorithm. In volume compression, however, we usually have $\mathbf{B}_{(n)} \in \mathbb{R}^{I \times J}$ with $I \ll J$. Since we do not need the J right singular vectors, in such cases it is much more efficient to compute first the matrix $\widehat{\mathbf{B}}_{(n)} = \mathbf{B}_{(n)} \cdot \mathbf{B}_{(n)}^T$ and then obtain all left singular vectors $\mathbf{U}^{(n)}$ from the full eigenvalue decomposition $\widehat{\mathbf{B}}_{(n)} = \mathbf{U}^{(n)} \mathbf{\Lambda}^{(n)} {\mathbf{U}^{(n)}}^T$. Since $\widehat{\mathbf{B}}_{(n)}$ is a real symmetric matrix, its eigenvalue diagonalization always exists and we can use a more efficient specialized solver. The remaining rightmost part of the SVD follows from

$$\mathbf{\Sigma}^{(n)} {\mathbf{V}^{(n)}}^T = (\mathbf{U}^{(n)})^{-1} \mathbf{B}_{(n)} = {\mathbf{U}^{(n)}}^T \mathbf{B}_{(n)} \tag{7}$$

as the factor matrix $\mathbf{U}^{(n)}$ is orthogonal. This process is undertaken N times. In the last iteration we reshape (fold) $\mathbf{\Sigma}^{(N)} \cdot {\mathbf{V}^{(N)}}^T$ back into an N-dimensional tensor, namely the core $\mathcal{B}$.

4.2 Bit-plane Coding

Once the Tucker core $\mathcal{B}$ is available we can turn to our coefficient coding scheme. Note that, since no truncation was performed, we have not yet incurred any loss of accuracy other than floating-point round-off errors. Our goal now is to produce an approximate core $\widetilde{\mathcal{B}}$ such that its SSE satisfies

$$\text{SSE}(\mathcal{B}, \widetilde{\mathcal{B}}) = \|\mathcal{B} - \widetilde{\mathcal{B}}\|^2 \leq s, \tag{8}$$

where s is a user-defined bound (recall that, due to factor orthogonality as we saw in Sec. 3.1, compression error is directly related to the error in the core coefficients). We address this via *bit-plane coding* in the spirit of EZW [34], SPIHT [35], or EBCOT [36]. We start off by scaling each coefficient's absolute value into a 64-bit unsigned integer

$$c \mapsto \left\lfloor |c| \cdot 2^{63 - \lfloor \log_2(m) \rfloor} \right\rfloor, \tag{9}$$

where $m := \max_{c \in \mathcal{B}} \{|c|\}$ is the core's largest element (in absolute value). The signs are dealt with separately; see later in this section. Each integer as given by Eq. 9 has a decomposition in powers of 2: $2^{63} \cdot c_{63} + ... + 2^0 \cdot c_0$ and defines a row of our binary matrix $\mathbf{M}$. Since in IEEE 754 every 64-bit floating-point number uses at most 53 significant bits, each row of $\mathbf{M}$ will have at least 11 zero bits. The basic principle that motivates bit-plane coding is the fact that for any bit plane p, all bits in the column $\mathbf{M}[:, p]$ are equally important. We propose a greedy encoding strategy: we transmit first all bits in the most-significant bit plane $p = 63$. We then move on to the next plane, i.e. $p := p - 1$, and repeat. We encode each column from top to bottom and terminate as soon as we fall below the given SSE error tolerance s, which usually means that we encode only the top portion of the last column. Note that, instead of error, an alternative stopping criterion based on limiting the compressed file size could be similarly devised and the compression process stopped accordingly.

Since the binary matrix is usually sparse, we initialize it to 0. The error is largest in the beginning and decreases every time a 1 bit is transmitted. This approach gives the same importance to all bits that lie within the same bit plane, and it ensures that the error it introduces is no larger than the prescribed target SSE. We have chosen a lossless compression strategy to process all selected bits (that is, up to the threshold breakpoint at plane P). Statistically, we can expect this to yield high compression ratios thanks to the massive imbalance between the number of 0 and 1 bits in most leading bit planes (see examples in Figs. 4 and 5). Furthermore, long strings of consecutive zero bits (*runs*) happen frequently, and their lengths have a low-entropy distribution. See Fig. 6 for an example; we encode each sequence of k 0-bits that is followed by a 1 (or ends the column) as the integer k. For example, the binary string `01110001` becomes $[1, 0, 0, 3]$.

Fig. 7 shows theoretical coding performance when storing each bit plane's sequence of zero-run lengths. We plot the bit rate (number of bits after compression, divided by bits before compression) that an ideal entropy coder would need for the given set of integer symbols in the RLE, without accounting for storing a table of frequencies. The bit rate is computed as $\sum_i f_i \cdot \log_2(n/f_i)$, where f_i counts how many times the i-th symbol occurs and $n = \sum_i f_i$ is the total number of symbols to transmit. Modern entropy coders (Huffman and, especially,

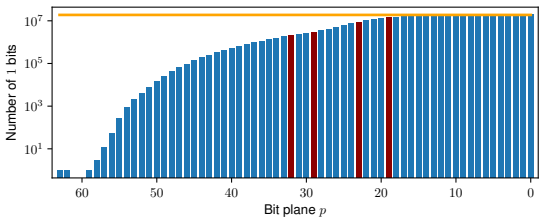


Fig. 4. Number of 1 bits for each bit plane in the HOSVD transform of the Density data set (see Sec. 6). The orange line is set at half the total number of core coefficients $C/2$. The threshold plane P needed for 1000:1, 300:1, 100:1, and 50:1 compression is shown for each case from left to right in red.

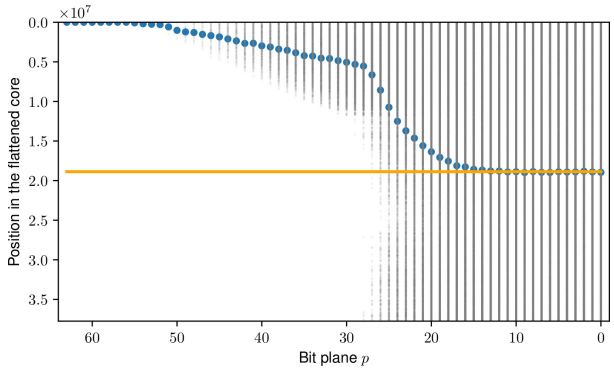


Fig. 5. Density plot showing all 1 bits (gray dots) for all bit planes in the flattened HOSVD transform of the Density volume. The center of mass of each set of points is shown as a blue dot. The orange line is set at the center $C/2$.

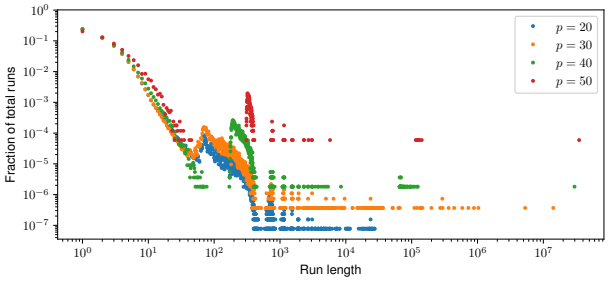


Fig. 6. Distribution of run lengths for four different bit planes (Density volume). Note how unbalanced the frequencies are and how they tend to concentrate around a few specific regions along the x-axis.

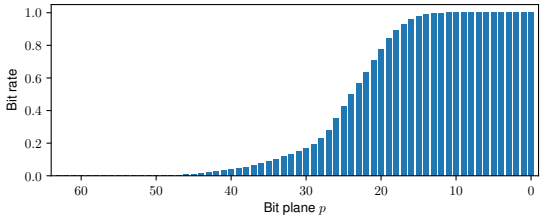


Fig. 7. Bit rate that would be achieved by a perfect entropy coder compressing all zero run lengths within each bit plane $p = 63, \ldots, 0$ (Density volume). No compression is possible for $p \leq 12$.

arithmetic coding) are usually very close to this information-theoretical optimum.

Our statistical analyses on the columns of $\mathbf{M}$ motivate us to handle each coefficient's *leading bits* (i.e. its leftmost 1 bit and all 0 bits on its left) differently from its *trailing bits* (i.e. 0 or 1 bits that appear to the right of the leftmost 1):

- Leading bits tend to form long runs of zeros with very low entropy along the columns of $\mathbf{M}$. We compress them without loss via RLE followed by AC.
- Trailing bits are close to being uniformly random; RLE+AC cannot compress them well. We thus store them verbatim, which is naturally faster.

Since most planes use a combination of both coding methods, we need an efficient data structure to keep track of leading vs. trailing bits, i.e. a *significance map*. As we work our way from the leftmost $b = 63$ towards less significant planes we update a binary mask $\mathcal{M}$ that records all coefficients that have already become significant (their leftmost 1 bit has been encountered). The mask starts empty and gains members progressively. See Fig. 8 for a toy illustration.

Remarks Recall that this algorithm concerns absolute values only. Like trailing bits, the signs of significant coefficients are close to uniformly random, and we transmit them verbatim as well.

The final compression ratio may vary if one chooses FORTRAN-ordering (left-to-right) instead of C-ordering when flattening the core, since all bit planes will contain different orderings of `0` and `1` bits. We found this to influence very little the overall compressed file size in practice.

4.3 Factor Compression

The square factors $\{\mathbf{U}^{(n)}\}_n$ usually account for a small proportion of the overall number of elements in a full HOSVD decomposition, e.g. $(3 \cdot 256^2)/256^3 \approx 1\%$ for a 256^3-sized volume. However, this can become a significant overhead if the factors are not compressed as carefully as the core (Sec. 4.2). Although factor matrix compression is an important part of a Tucker-driven compression pipeline, previous related approaches [2, 22, 23] did not place a particular emphasis on it.

To encode the factors we essentially reuse the same compression algorithm that we proposed for the core. Nonetheless, two important details deserve special consideration. First, factor matrix columns have vastly different importances: each factor column interacts with one core slice only, and such slices have varying norms (recall Eq. 5). In practice, those norms are orders of magnitude apart (see Fig. 9), and a proper weighting of our factor columns is in order. Recall (Sec. 4.1) that the factors contain the left singular vectors of an SVD decomposition: $\mathbf{B}_{(n)} = \mathbf{U}^{(n)} \cdot \mathbf{\Sigma}^{(n)} \cdot \mathbf{V}^{(n)^T}$, where $\mathbf{B}_{(n)}$ is our tensor reconstructed along dimension n only. The matrix $\mathbf{\Sigma}^{(n)}$ is diagonal and holds the core slice norms $\sigma_1^{(n)}, \ldots, \sigma_{I_n}^{(n)}$. Since $\mathbf{V}^{(n)}$ is orthogonal, any SSE error on $\mathbf{U}^{(n)} \cdot \mathbf{\Sigma}^{(n)}$ will produce the same SSE on $\mathbf{B}_{(n)}$. In other words, in order to control the error that is introduced due to the n-th factor we need to compress $\mathbf{U}^{(n)} \cdot \mathbf{\Sigma}^{(n)}$. Simply put, we multiply each j-th column of $\mathbf{U}^{(n)}$ by its corresponding core slice norm $\sigma_j^{(n)}$ prior to compression. Since those norms account for a small set of floating point values, we afford to store them explicitly as part as our compression so that the procedure is efficiently reversible for the decompression.

The second question is how many bits we should allocate for each of the factor matrices. Stopping during the same bit plane threshold P that we determined for the core would be a clearly suboptimal choice: even though every factor is just as important as the core for the overall error, they have far fewer elements. Thus, it is reasonable to spend more bits per factor coefficient than we did per core coefficient. We choose a cost-effective criterion that takes into account both compression ratio and quality. Consider the rate-distortion curve obtained by plotting the compression SSE error s_b vs. compressed file size S_b after encoding each core bit $b = 1, \ldots, 64C$. In the beginning we spend zero bits for compression and the error is maximal. The first bits are very *cheap* to encode (they are mostly zeros), yet decrease the error greatly since they belong to the most significant bit planes. In other words, the ratio $\alpha_b := \Delta s_b / \Delta S_b = (s_b - s_{b-1})/(S_b - S_{b-1})$ is large for $b \ll 64C$. However, that ratio decreases as more bits are transmitted:

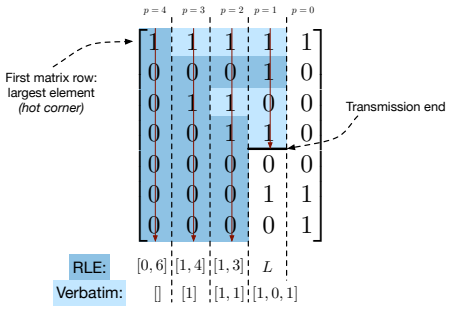


Fig. 8. Simplified example coding of 7 coefficients at $P = 5$ bits each. Encoded bits are highlighted in dark and light blue; the order is shown by the red arrows, left to right. The transmission was stopped based on a threshold in bit-plane $P = 1$. For each coefficient, its leftmost 1 and all leading 0's are compressed using RLE+AC (dark blue), whereas trailing bits are stored verbatim (light blue). The significance mask went from zero members at $p = 4$ to three at $p = 1$.

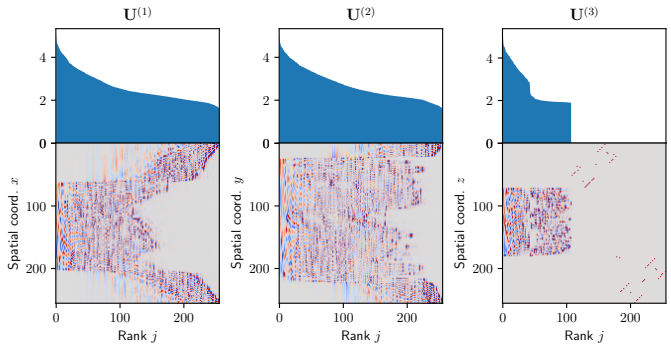


Fig. 9. The three factor matrices obtained by decomposing the Engine data set. On top we show, in $\log_{10}$ scale, the corresponding core slice norm $\sigma_j^{(n)}$ of every j-th column of each n-th factor. Note that, as a consequence of large empty background regions in the volume, many core slices along the third mode are zero.

average entropy increases, whereas the bit planes lose significance as p decreases. We estimate the ratio α_b achieved at the core transmission breakpoint b, and use it as stopping criterion: we halt the encoding of each n-th factor at the first bit $b^{(n)}$ such that $\alpha_{b^{(n)}} \leq \alpha_b$. By harmonizing all stopping criteria on a single α_b, our strategy ensures that a reasonable price is paid in all cases. As we expected, it does result in more bit planes being used than those selected for the core. We also observed that, the smaller the factor matrix, the more bit planes we can generally afford before surpassing α_b.

5 Decompression and Post-processing

Decompression follows straightforwardly by inverting the steps described above. HOSVD core bit planes are decompressed in the same order as they were transmitted, the factors are then decompressed, and finally the HOSVD transform is reversed via N TTM products as in Eq. 2. We speed the TTMs up by detecting and discarding core slices that have become zero during compression. In order to reverse the mixed RLE+AC/verbatim we again use an incrementally-updated mask of significant coefficients similarly to Sec. 4.2. Decompression is significantly faster than compression (see Sec. 6) since no covariance and eigenvalue decomposition are needed. After decompression, we apply proper rounding to the core and factors' coefficients. We assume that the residual (i.e. error between the original coefficient and the approximate one) follows an approximately uniform distribution $[0, 2^P - 1]$. Instead of simply assuming that the least significant bits $p < P$ of a coefficient are zero, we take the expected value 2^{P-1}.

Compression-domain Resampling

Thanks to multilinearity, filtering operations on compressed tensors can be efficiently performed via convolution on their factor matrices; see e.g. [37, 38]. Separable filters $\mathcal{F} = \boldsymbol{u}^{(1)} \otimes \cdots \otimes \boldsymbol{u}^{(N)}$ are particularly straightforward to apply:

$$\mathcal{T} * \mathcal{F} = \mathcal{B} \times_1 (\mathbf{U}^{(1)} * \boldsymbol{u}^{(1)}) \times_2 \cdots \times_N (\mathbf{U}^{(N)} * \boldsymbol{u}^{(N)}) \qquad (10)$$

where $\mathbf{U}^{(n)} * \boldsymbol{u}^{(n)}$ denotes column-wise convolution between a matrix and a column vector. In other words, each row of the n-th factor becomes a linear combination of its neighboring rows, weighted by the vector $\boldsymbol{u}^{(n)}$. This operation has a negligible cost compared to the decompression, which has to be performed anyway for visualization. Following this principle, we have implemented three options for compressed-domain decimation:

- Downsampling: we simply select an evenly spaced subset of the factor rows and discard the rest.
- Box filtering: we average consecutive rows together.
- Separable Lanczos-2: we convolve column-wise the factors with a 1D Lanczos kernel prior to subselection. We use the 3-lobed kernel, i.e. 5 samples with window parameter equal to 2:

$$u(x) = \begin{cases} \operatorname{sinc}(x) \cdot \operatorname{sinc}(x/2) & \text{if } -2 < x < 2 \\ 0 & \text{otherwise} \end{cases}$$

with $x = \{-2, 1, 0, 1, 2\}$, where $\operatorname{sinc}(x) := \frac{\sin(\pi x)}{\pi x}$.

In our implementation the user can specify index ranges and strides via NumPy-style notation. Immediate applications include previewing, subvolume selection and slicing, reversing dimensions, frame-by-frame visualization in time-dependent data, etc. In all these cases the Tucker core remains unchanged, so the filtering and downsampling asymptotic costs amount to only $O(NI^2 \log I)$ operations for the column-wise factor convolution where $I := \max\{I_1, \ldots, I_N\}$.

6 Results

We tested the proposed method with 12 integer and floating-point volume data sets, along with two time-varying volumes (all details and sources are shown in Tab. 1). We use Eigen 3.2.9 for matrix manipulations, products and eigenvalue decomposition, more specifically its `SelfAdjointEigenSolver` class for symmetric real matrices. We used a 4-core Intel i7-4810MQ CPU with 2.80GHz and 4GB RAM. All renderings were generated via volume ray casting in ParaView [39].

We have measured the compression performance of TTHRESH against four state-of-the-art algorithms:

- Tucker rank truncation and fixed core quantization [23] (our own implementation). We use 8 and 32 bits for core and factor coefficients, respectively, and label this algorithm as TRUNC.
- ZFP [17] (version 0.5.4 as implemented in [44]). We use its fixed accuracy mode (which usually yields the best compression rates), serial execution mode, and vary its absolute error tolerance (`-a`).
- SZ [4] (version 2.0.1.0 as implemented in [45]). We use the relative error bound mode and vary accordingly the relative bound ratio parameter (`relBoundRatio`).
- SQ [6] (our own implementation). We vary the absolute error tolerance and stream the output through the LZMA lossless compressor as advised in the original paper.

All codes were compiled with `g++` at maximum optimization (`-O3` flag). Since SZ does not readily support integer data types, we first cast all 8-bit volumes to 64-bit floats; we measure compression ratios w.r.t. the original data for all five compressors. Figs. 10 and 11 show

Table 1. The 14 data sets tested in this paper.

Name	Dimensions	Type	Size	Source
"Foot", "Engine"	$256 \times 256 \times 256$	8-bit unsigned int	16 MB	The Volume Library [40]
"Teapot"	$256 \times 256 \times 178$	8-bit unsigned int	11.1 MB	The Volume Library [40]
"Isotropic-coarse", "Isotropic-fine", "Channel", "MHD", "Mixing"	$512 \times 512 \times 512$	32-bit float	512 MB	All available pressure fields from the Johns Hopkins Turbulence Database [1]
"Viscosity", "Density"	$384 \times 384 \times 256$	64-bit float	288 MB	Lawrence Livermore National Laboratory (Miranda simulation [41])
"U"	$288 \times 192 \times 28$	64-bit float	11.8 MB	National Center for Atmospheric Research (Community Earth System Model [42])
"Jet-u"	$400 \times 250 \times 200$	64-bit float	152.6 MB	Sandia National Laboratories (S3D simulation [43])
"Isotropic-fine-time"	$64 \times 64 \times 64 \times 64$	32-bit float	64 MB	Time-varying version of the Isotropic-fine (third row)
"Hurricane"	$50 \times 50 \times 91 \times 48$	32-bit float	42 MB	SciVis 2004 Contest Data Set, QVAPOR field

the resulting error curves in terms of PSNR vs. compression ratio over all sample data sets. We observe a recurring pattern from lower to higher compression ratio: our proposed algorithm performs similarly (sometimes worse) than other methods for lower ratios, up to a tipping point after which it is better by a widening margin. Although this point can vary significantly, the general behavior is consistent across all data sets we tested. We argue that the usual rates at which TTHRESH performs best are the most adequate for visualization purposes.

To support this claim we present several volume renderings before and after compression at two levels of quality in Fig. 12 as well as in the paper teaser (Fig 1).

We observe how the fixed number of quantization bits used by TRUNC entails a fixed error that often dominates that introduced by the rank truncation. This explains the flat PSNR curves for TRUNC over several data sets in Figs. 10 and 11. Interestingly, SQ is sometimes not monotonic (see e.g. the results for the Channel volume). We attribute this to its set partitioning strategy [6], whose resulting partitions can be highly sensitive to even small variations of the error tolerance specified.

We also note that, at medium to high compression ratios, TTHRESH tends to preserve well the coarsest features and smoothen out or eliminate smaller details. See for example Fig. 13 for a sequence of zoomed-in renderings under progressively heavier compression that make this phenomenon evident. It is only at exceedingly high ratios that block-like features start to appear, whereas other algorithms suffer from a much faster visual degradation. This multiscale feature-selective behavior is similar to that observed in truncation-based tensor compression [22].

It is further backed by empirical observation of the Fourier spectra of our compressed data sets. In Fig. 14 we show three example factor matrices for the U volume and their corresponding Fourier transforms along the spatial dimension (i.e. factor columns). Often, each vector in the HOSVD basis corresponds roughly to one frequency wavefunction that has been tuned to better match the specific input data set. Due to the *hot corner* phenomenon, most insignificant HOSVD core coefficients are those that correspond to trailing factor columns (recall also Fig. 9) that, in light of Fig. 14, contain mostly high frequencies. In short, we can expect our coder to act as a low-pass filter. This is consistent with what we showed in Fig. 13, namely smooth low-frequency artifacts that arise in data compressed with TTHRESH.

To better illustrate this shift towards low frequencies, we depict in Fig. 15 the Fourier magnitude histograms obtained at different compression rates. Note that SZ, SQ and ZFP behave in the opposite way as they rather shift the spectrum towards the high-frequency end.

Regarding computational speed, we plot in Fig. 16 the compression and decompression times for the smallest data set (the Teapot, 11.1MB) as well as for one of the 512MB ones (the Isotropic-fine). Our method is between 0.5 and 2 orders of magnitude slower than the fastest one, namely ZFP, but compression is generally faster than SQ. It is rather asymmetric as we expected from Sec. 5: the average compression/decompression times for those measurements was 2.4s/1.0s (Teapot) and 61.5/25.8s (Isotropic-fine). To give more insight on the differences between compression and decompression costs, we have broken them down in Fig. 17 (Teapot volume). Note also that the varying accuracy curves between all five compared algorithms make a fully fair comparison difficult. For consistency with Fig. 10 we did the comparison in terms of time vs. compression ratio, but note that TTHRESH fares better in terms of quality in large parts of the error spectrum.

Our last experiment is reported in Fig. 18, where we demonstrate visual results of decimation along the factor matrices (Sec. 5) followed by decompression. Note the differences between the three methods implemented and the superiority of Lanczos' kernel for this task.

7 Discussion

We observe that the proposed algorithm achieves competitive accuracy at low to medium compression ratios and consistently outperforms other compressors at medium to high ratios; see the higher PSNR curves for our method in the highlighted regions of each plot from Fig. 10. The overtaking point at which TTHRESH surpasses the other algorithms (marked by the vertical dotted lines) typically produces renderings that are already close to visually indistinguishable to the original data set. This is especially true for higher bit depths. We believe our method is thus a good choice for applications with reasonable error tolerance (chiefly, visualization-related). In addition, we showed how our choice of global bases helps the method achieve a very smooth degradation rate. This is manifested both as a Fourier spectrum shift and as a visually parsimonious erosion of the smaller details and features. Since the transmission can be stopped at any arbitrary point within any bit plane, the range of possible final errors has a very fine granularity. Also, the error that arises from the core compression is upper-bounded by definition of the stopping criterion. Last, the compressed-domain filtering and resampling features are rather unique strengths of the tensor decomposition framework, only possible thanks to its multilinearity. Any separable filter and resampling can be applied with little cost by manipulating the factors column-wise before the final Tucker reconstruction. This is often much more challenging in other compression methods, especially brick-based and non-transform ones. Even though Lanczos antialiasing results are visually superior when lowering a data set's resolution, we believe the other decimation methods remain useful for other operations such as region/slice selection, projections, etc.

Limitations

TTHRESH's compression rates and smooth degradation come at the price of its monolithic approach to the transform core. This puts it in the slower end of the spectrum of volume compressors, especially compared to those designed for speed such as ZFP. Random-access decompression is also relatively costly, as one must traverse the whole core in all cases. To improve compression/decompression speed one may resort to splitting the data set and using the proposed compressor

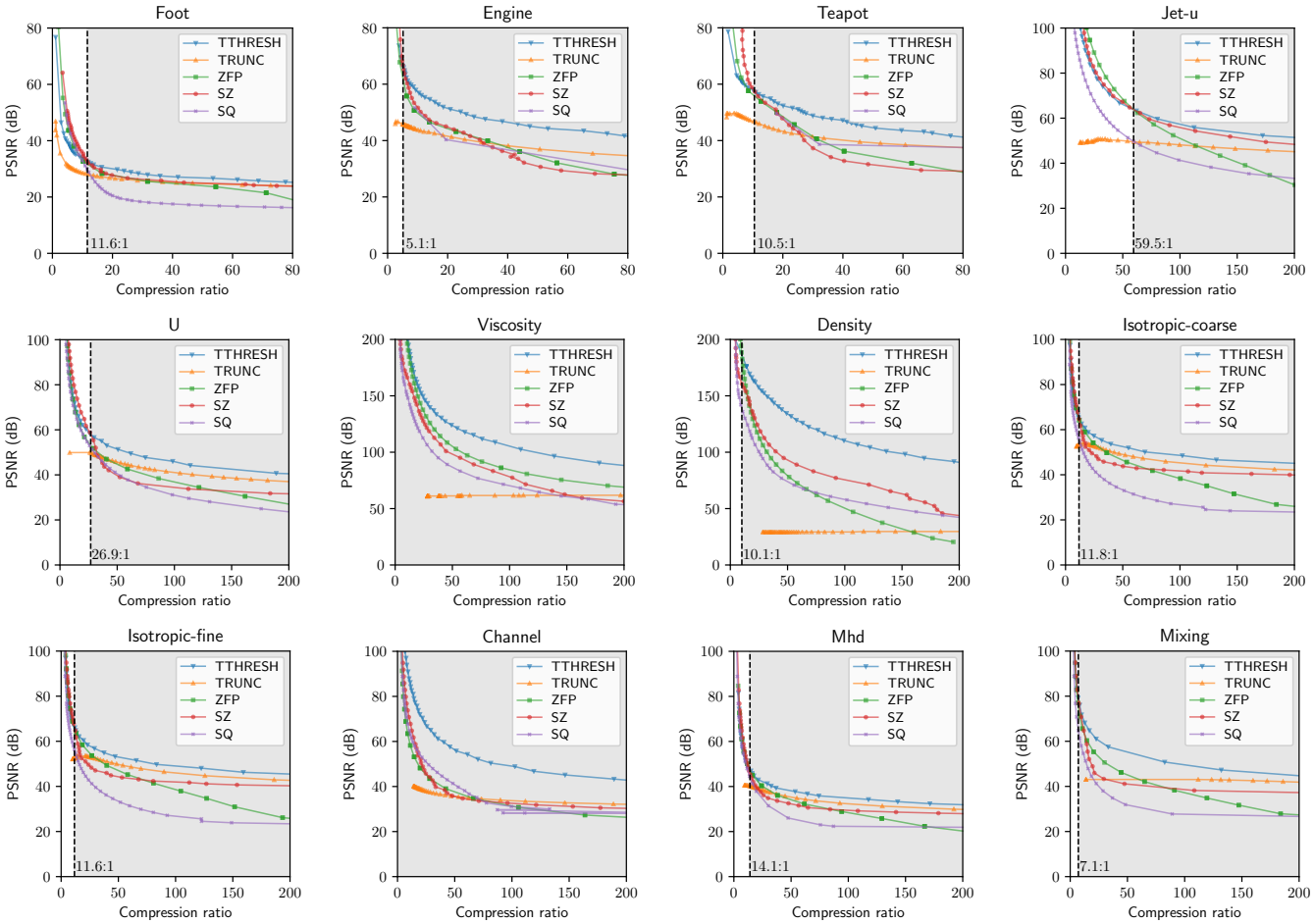


Fig. 10. Compression quality curves (higher is better) for our method compared to TRUNC, ZFP, SZ, and SQ over 12 example volumes and varying compression ratios (up to 80:1 for integer data, and 200:1 for floating-point data). We show in gray all ratios where TTHRESH offers the highest PSNR among all compressors.

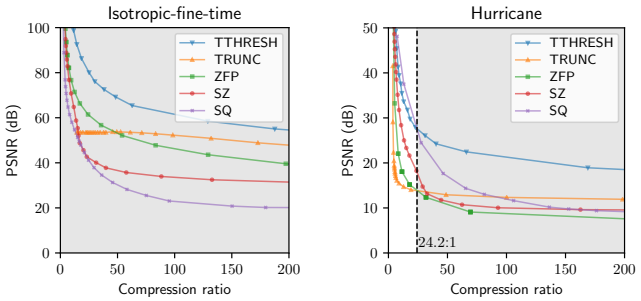


Fig. 11. Compression quality curves for two time-varying volumes; see also Fig. 10.

on a brick-by-brick basis, in the spirit of tensor-compressed multiresolution rendering systems [23, 38].

8 Conclusion

We have introduced a novel tensor decomposition based compression algorithm with an emphasis on storage/visualization applications whose foremost priority is data reduction at high compression ratios. Unlike previous HOSVD-driven approaches, this reduction is achieved by keeping all ranks followed by careful lossless compression of all bit planes up to a certain threshold. It is, to the best of our knowledge, the first tensor compressor (and specifically, HOSVD-based) that uses a bit-plane based strategy, also on the factor matrices. The main property we exploited was factor orthogonality, which ensures that all coefficients affect equally the final l^2 error and so allows us to sort the full core as a single block. Our algorithm possesses advantages that are inherent to multilinear transforms in general and tensor decompositions in particular, including support for linear manipulation of the data set in the compressed domain.

We developed TTHRESH focusing primarily on optimizing data reduction rates, and less so on general compression/decompression speed. We have realized that these speeds (especially compression) can be increased significantly at a relatively small accuracy cost in multiple ways, for example by moderating the eigensolver's number of iterations or by preemptively discarding some of the least important core slices. Also, we note that progressive decompression is compatible with the proposed coder: after all, we encode bit planes from more to less significant. To achieve progressiveness, coefficient signs should be encoded as soon as the coefficient becomes significant, e.g. using a negabinary base or deferred sign coding. In addition, factor columns should be encoded as soon as they are needed. These possibilities will be the subject of future investigation.

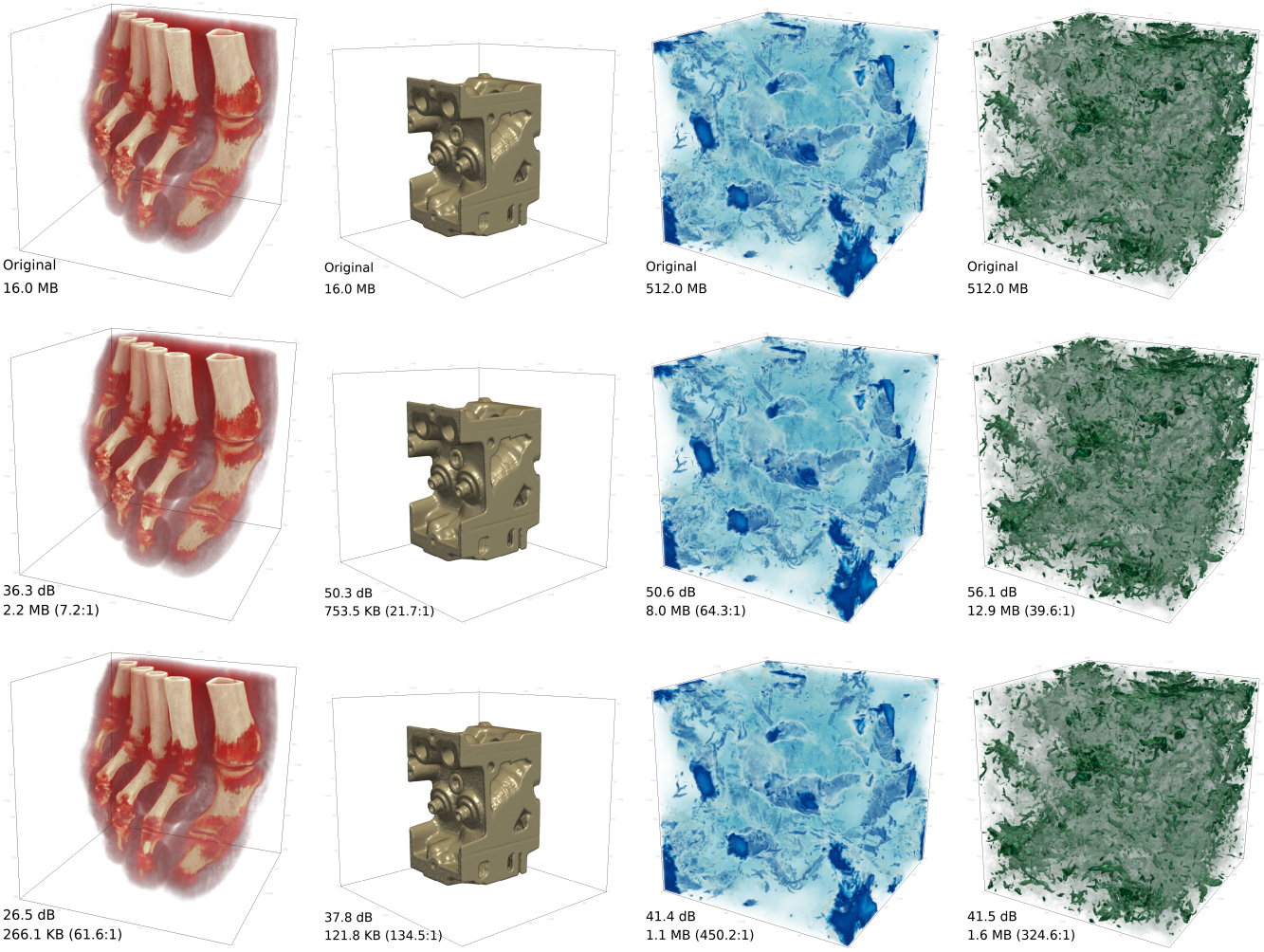


Fig. 12. Four example volumes: the Foot and Engine CT scans (both 8-bit unsigned int), and the Isotropic-coarse and Mixing turbulence simulations (both 32-bit float). Rows from top to bottom: original, higher quality, and lower quality. All these compressed volumes take half or less the space needed with the other four methods tested at equivalent PSNR (except the Foot at higher quality, which performs similarly); see also Fig. 10.

ACKNOWLEDGMENTS

This work was partially supported by the University of Zurich's Forschungskredit "Candoc", grant number FK-16-012, and partially performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. The authors wish to thank Stephen Hamilton from Johns Hopkins University as well as the other institutions listed in Tab. 1 for kindly providing the data sets we have used for testing.

REFERENCES

[1] "Johns Hopkins Turbulence Database," http://turbulence.pha.jhu.edu/newcutout.aspx.

[2] R. Ballester-Ripoll and R. Pajarola, "Lossy volume compression using Tucker truncation and thresholding," *The Visual Computer*, pp. 1–14, 2015.

[3] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. F. Samatova, "Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data," in *Euro-Par Conference on Parallel Processing*, vol. 1, Aug. 2011, pp. 366–379.

[4] S. Di and F. Cappello, "Fast error-bounded lossy HPC data compression with SZ." in *International Parallel and Distributed Processing Symposium*, May 2016, pp. 730–739.

[5] M. Soler, M. Plainchault, B. Conche, and J. Tierny, "Topologically controlled lossy compression," in *IEEE PacificVis Symposium*, 2018, pp. 46–55.

[6] J. Iverson, C. Kamath, and G. Karypis, "Fast and effective lossy compression algorithms for scientific datasets," in *Euro-Par Conference on Parallel Processing*, 2012, pp. 843–856.

[7] E. Gobbetti, J. Iglesias Guitián, and F. Marton, "COVRA: A compression-domain output-sensitive volume rendering architecture based on a sparse representation of voxel blocks," *Computer Graphics Forum*, vol. 31, no. 3, pp. 1315–1324, 2012.

[8] S. Guthe and M. Goesele, "Variable length coding for GPU-based direct volume rendering," in *Vision, Modeling and Visualization*, October 2016.

[9] M. Balsa Rodriguez, E. Gobbetti, J. A. Iglesias Guitián, M. Makhinya, F. Marton, R. Pajarola, and S. K. Suter, "A survey of compressed GPU direct volume rendering," Eurographics State of The Art Report (STAR), May 2013.

[10] B.-L. Yeo and B. Liu, "Volume rendering of dct-based compressed 3d scalar data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, no. 1, pp. 29–43, March 1995.

[11] S. Muraki, "Volume data and wavelet transforms," *IEEE Computer Graphics and Applications*, vol. 13, no. 4, pp. 50–56, July 1993.

[12] S. Guthe and W. Strasser, "Real-time decompression and visualization of animated volume data," in *Proceedings IEEE Visualization*, Oct 2001, pp. 349–572.

[13] K. G. Nguyen and D. Saupe, "Rapid high quality compression of volume data for visualization," *Computer Graphics Forum*, vol. 20, no. 3, pp. 49–57, 2002.

[14] S. Guthe, M. Wand, J. Gonser, and W. Strasser, "Interactive rendering of large volume data sets," in *Proceedings IEEE Visualization*, Oct 2002, pp.

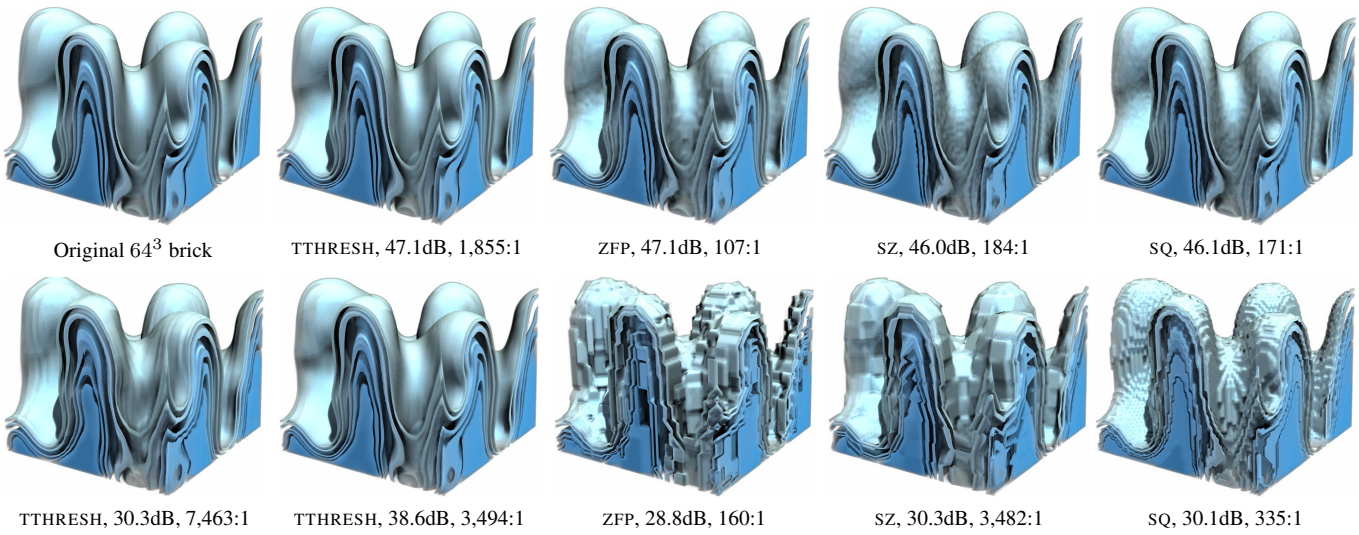
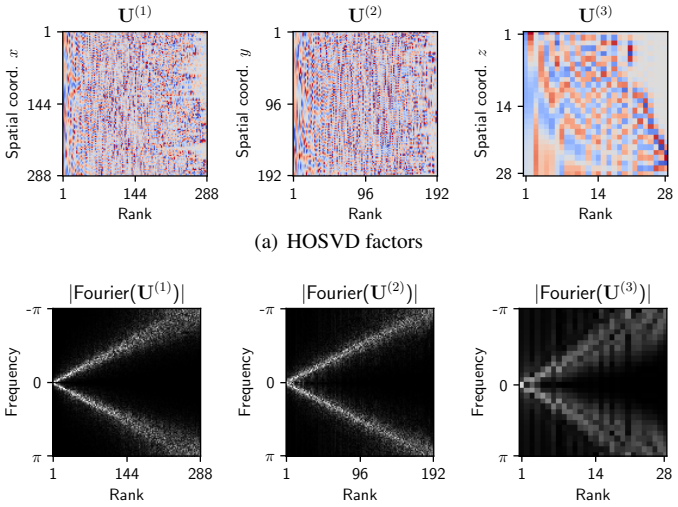


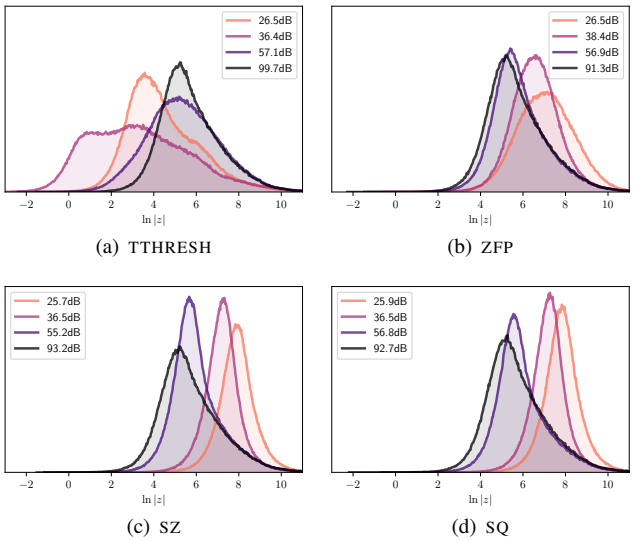
Fig. 13. The HOSVD produces custom data-dependent bases that make the proposed algorithm degrade visually very smoothly up to extreme compression rates. Depicted is a 64^3 brick that was cut out from the center of the Density volume after compression with varying quality and algorithms (measurements correspond to the full volume). Unlike other compressors, TTHRESH avoids blocky artifacts; instead, it erodes and merges features at progressively coarser scales.



(a) HOSVD factors

(b) Their column-wise Fourier transform, in magnitude

Fig. 14. Factors obtained from the U data set along with their Fourier transform. HOSVD bases often resemble cosine wavefunctions and are rather sparse signals in the frequency domain.



(a) TTHRESH (b) ZFP

(c) SZ (d) SQ

Fig. 15. Logarithmic histograms of the Fourier transform magnitude at different compression quality levels of the U volume. TTHRESH shifts the spectrum towards lower frequencies, as opposed to the other three methods which tend to introduce higher frequencies instead.

53–60.

[15] X. Wu and T. Qiu, "Wavelet coding of volumetric medical images for high throughput and operability," *IEEE Transactions on Medical Imaging*, vol. 24, no. 6, pp. 719–727, June 2005.

[16] J. Clyne, P. Mininni, A. Norton, and M. Rast, "Interactive desktop analysis of high resolution simulations: Application to turbulent plume dynamics and current sheet formation," *New Journal of Physics*, vol. 9, no. 8, p. 301, 2007.

[17] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2674–2683, 2014.

[18] H. Wang and N. Ahuja, "Compact representation of multidimensional data using tensor rank-one decomposition," in *Proceedings Pattern Recognition Conference*, 2004, pp. 44–47.

[19] Q. Wu, T. Xia, and Y. Yu, "Hierarchical tensor approximation of multi-dimensional images," in *Proceedings IEEE International Conference in Image Processing*, vol. 4, 2007, pp. 49–52.

[20] Q. Wu, T. Xia, C. Chen, H.-Y. S. Lin, H. Wang, and Y. Yu, "Hierarchical tensor approximation of multidimensional visual data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 1, pp. 186–199, 2008.

[21] S. K. Suter, C. P. Zollikofer, and R. Pajarola, "Application of tensor approximation to multiscale volume feature representations," in *Proceedings Vision, Modeling and Visualization*, 2010, pp. 203–210.

[22] S. K. Suter, J. A. Iglesias Guitián, F. Marton, M. Agus, A. Elsener, C. P. Zollikofer, M. Gopi, E. Gobbetti, and R. Pajarola, "Interactive multiscale tensor reconstruction for multiresolution volume visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2135–2143, 2011.

[23] S. K. Suter, M. Makhinya, and R. Pajarola, "TAMRESH: Tensor approximation multiresolution hierarchy for interactive volume visualization," *Computer Graphics Forum*, 2013.

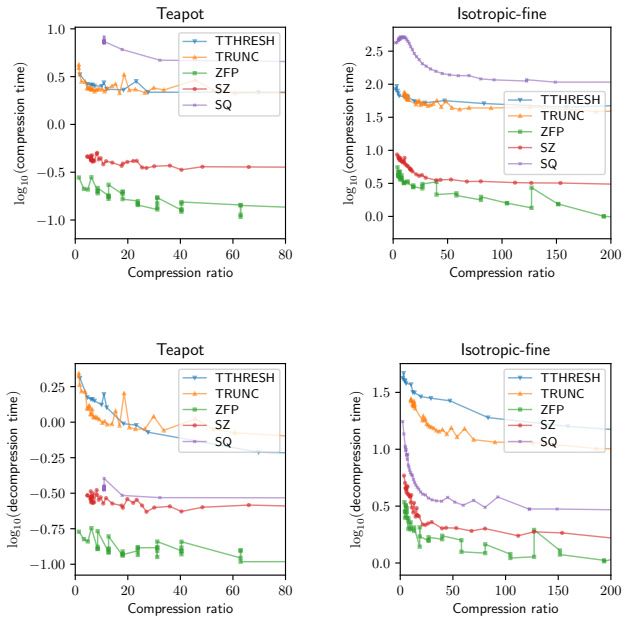
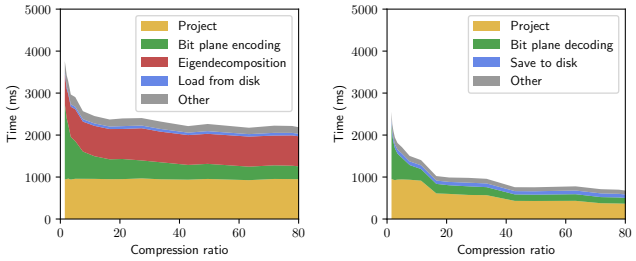


Fig. 16. Compression (top row) and decompression (bottom row) times (in seconds) for two volumes and a range of different compression ratios.



(a) Compression (b) Decompression

Fig. 17. Compression/decompression times (Teapot): loading/saving the data set, eigenvalue decomposition, tensor projection with Tucker factors, and bit plane processing.

[24] R. Ballester-Ripoll, S. K. Suter, and R. Pajarola, "Analysis of tensor approximation for compression-domain volume visualization," *Computers and Graphics*, vol. 47, pp. 34–47, 2015.

[25] G. Wetzstein, D. Lanman, M. Hirsch, and R. Raskar, "Tensor displays: Compressive light field synthesis using multilayer displays with directional backlighting," *ACM Transactions on Graphics*, vol. 31, no. 4, pp. 80:1–11, 2012.

[26] R. Ballester-Ripoll and R. Pajarola, "Tensor decompositions for integral histogram compression and look-up," *IEEE Transactions on Visualization and Computer Graphics*, vol. PP, pp. 1–12, 2018.

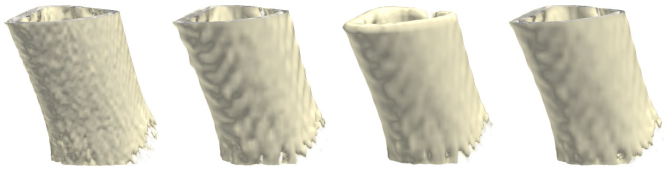
[27] R. Ruiters and R. Klein, "BTF compression via sparse tensor decomposition," *Computer Graphics Forum*, vol. 28, no. 4, pp. 1181–1188, 2009.

[28] Y.-T. Tsai, "Parametric representations and tensor approximation algorithms for real-time data-driven rendering," Ph.D. dissertation, National Chiao Tung University, May 2009.

[29] Y.-T. Tsai and Z.-C. Shih, "K-clustered tensor approximation: A sparse multilinear model for real-time rendering," *ACM Transactions on Graphics*, vol. 31, no. 3, pp. 19:1–19:17, 2012.

[30] Y.-T. Tsai, "Multiway K-clustered tensor approximation: Toward high-performance photorealistic data-driven rendering," *ACM Transactions on Graphics*, vol. 34, no. 5, pp. 157:1–15, 2015.

[31] L. de Lathauwer, B. de Moor, and J. Vandewalle, "On the best rank-1 and rank-(R_1, R_2, ..., R_N) approximation of higher-order tensors," *SIAM Journal of Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1324–1342, 2000.



(a) Original (b) Downsampling (c) Box filter (d) Lanczos-2

Fig. 18. (a) a 60^3 region of the Foot data set. (c-d): 2-fold decimated versions at 7.5:1 compression using the three different methods from Sec. 5. Lanczos minimizes both the blocky aliasing along edges of pure downsampling and the erosion that the box filter incurs (top of the bone).

[32] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[33] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.

[34] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3445–3462, Dec 1993.

[35] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 243–250, June 1996.

[36] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Transactions on Image Processing*, vol. 9, no. 7, pp. 1158–1170, July 2000.

[37] B. N. Khoromskij and V. Khoromskaia, "Low rank Tucker-type tensor approximation to classical potentials," *Central European Journal of Mathematics*, vol. 5, no. 3, pp. 523–550, 2007.

[38] R. Ballester-Ripoll, D. Steiner, and R. Pajarola, "Multiresolution volume filtering in the tensor compressed domain," *IEEE Transaction on Visualization and Computer Graphics*, to appear 2018.

[39] "ParaView: an open-source, multi-platform data analysis and visualization application," http://www.paraview.org.

[40] "IAPR-TC18 Data Sets," http://tc18.org/3D_images.html.

[41] W. H. Cabot and A. W. Cook, "Reynolds number effects on Rayleigh-Taylor instability with possible implications for type ia supernovae," *Nature Physics*, vol. 2, pp. 562 – 568, 2006.

[42] "Community Earth System Model by the National Center for Atmospheric Research," http://www.cesm.ucar.edu/index.html.

[43] R. Grout, A. Gruber, C. Yoo, and J. Chen, "Direct numerical simulation of flame stabilization downstream of a transverse fuel jet in cross-flow," *Proceedings of the Combustion Institute*, vol. 33, no. 1, pp. 1629 – 1637, 2011.

[44] "ZFP: Library for compressed numerical arrays," https://github.com/LLNL/zfp.

[45] "SZ: Error-bounded floating-point data lossy compressor," https://github.com/disheng222/SZ.