T4DT: Tensorizing Time for Learning Temporal 3D Visual Data

Mikhail Usvyatsov
mikhailu@ethz.ch

Rafael Ballester-Ripoll
rafael.ballester@ie.edu

Lina Bashaeva
lina.bashaeva@skoltech.ru

Konrad Schindler
schindler@ethz.ch

Gonzalo Ferrer
g.ferrer@skoltech.ru

Ivan Oseledets
l.Oseledets@skoltech.ru

1 Photogrammetry and Remote Sensing
ETH Zürich
Zürich, Switzerland

2 School of Science and Technology
IE University
Madrid, Spain

3 CAIT
Skolkovo Institute of Science and
Technology
Moscow, Russia

4 Artificial Intelligence Research Institute
Moscow, Russia

Abstract

Unlike 2D raster images, there is no single dominant representation for 3D visual data processing. Different formats like point clouds, meshes, or implicit functions each have their strengths and weaknesses. Still, grid representations such as signed distance functions have attractive properties also in 3D. In particular, they offer constant-time random access and are eminently suitable for modern machine learning. Unfortunately, the storage size of a grid grows exponentially with its dimension. Hence they often exceed memory limits even at moderate resolution. This work proposes using low-rank tensor formats, including the Tucker, tensor train, and quantics tensor train decompositions, to compress time-varying 3D data. Our method iteratively computes, voxelizes, and compresses each frame’s truncated signed distance function and applies tensor rank truncation to condense all frames into a single, compressed tensor that represents the entire 4D scene. We show that low-rank tensor compression is extremely compact to store and query time-varying signed distance functions. It significantly reduces the memory footprint of 4D scenes while remarkably preserving their geometric quality. Unlike existing, iterative learning-based approaches like DeepSDF and NeRF, our method uses a closed-form algorithm with theoretical guarantees.

1 Introduction

Recent advances in hardware development have made it possible to collect rich depth datasets with commodity devices. For example, modern AR/VR hardware can record videos of 3D data, thus capturing their temporal evolution to yield 4D datasets. Nevertheless, working
(a) Original, 284 GB (b) 1:12345, 24 MB (c) 1:793, 368 MB (d) 1:227, 1.24 GB

Figure 1: T4DT with different compression levels in OQTT format for a longshort-flying-eagle scene of resolution $512^3$ with 284 time frames. Only frames 1, 142, and 284 are depicted. The compression ratio is different from the actual memory consumption due to the padding of the time dimension to 512. High compression is achieved with $r_{max} = 400$, MSDM2 = 0.45 in Fig. 1b, medium compression with $r_{max} = 1800$, MSDM2 = 0.32 in Fig. 1c, and low compression / high quality with $r_{max} = 4000$, MSDM2 = 0.29 in Fig. 1d.

with 4D data presents computational challenges due to the curse of dimensionality. Furthermore, while 2D data mostly come in the form of raster images, there is an entire zoo of popular representations for 3D data, ranging from point clouds and meshes to voxel grids and implicit (neural) functions.

In this work, we address the case of representations based on regular grids. Such representations are highly structured and allow for efficient manipulation, e.g., they offer random access, slicing, and other operations in constant time. However, their storage requirements become a bottleneck: a grid of resolution $I$ along each dimension $D$ has $I^D$ elements. We propose applying tensor decompositions to leverage the high temporal and spatial correlation in grids that arise from time-evolving 3D data. Besides exploring the general idea of compression via low-rank constraints, we compare different tensor decomposition schemes and their potential for 4D video. For example, although the Tucker format [24, 25] is known to yield good results for the 3D case, it still suffers from the curse of dimensionality since it requires $O(r^D)$ elements w.r.t. the Tucker rank $r$, while $r$ must typically be chosen $\approx \frac{1}{\text{const.}}$ to achieve reasonable accuracy. Therefore, we explore hybrid low-rank formats based on the tensor train (TT) decomposition [17].

The Signed Distance Function (SDF, sometimes also called Signed Distance Field) and its truncated version, known as Truncated SDF or TSDF, is a convenient format for data fusion: several methods convert partial 3D views, even if initially in mesh format, to TSDFs to fuse them into complete models [6]. This pipeline is useful for multi-view stereo acquisition, laser scanners, as well as resource-limited devices (e.g., mobile augmented reality). TSDFs offer 1-to-1 correspondence across time, which is not easy to achieve with meshes if the objects undergo non-rigid deformation; while every mesh can be converted to a TSDF on the same voxel grid (with appropriately chosen resolution to avoid loss of detail). We experimentally analyze the effects of the low-rank constraint, using TSDF sequences of 3D scenes from the CAPE dataset [14, 19] and problem-specific quality metrics. Our method can compress time-varying grids to a usable size, while the same scenes in uncompressed form would each require hundreds of GigaBytes of memory.

In the following, Section 2 reviews applicable tensor methods and techniques for 3D and 4D data compression. Section 4 gives a detailed outline of our proposed approach, Section 5 demonstrates its the performance in an experimental study, followed by a discussion of limitations in Section 6, and concluding remarks in Section 7. Our contributions are:
1. The first tensor decomposition framework for compressing temporal sequences in 3D voxel space;
2. A benchmark and analysis of different decomposition schemes for both the spatial and temporal dimensions;
3. An open source implementation of our method based on the tntorch framework [26], available at https://github.com/prs-eth/T4DT.

2 Background and Related Work

In our work, we apply tensor methods to temporal sequences of 3D scenes. We consider grid representations of the 3D data, i.e., a tensor $A \in \mathbb{R}^{I_1 \times \ldots \times I_D}$ constitutes a discrete sampling of a $D$-dimensional space on a grid $\mathbb{I} = I_1 \times \ldots \times I_D$, with $I_d$ samples along dimension $d$.

2.1 SDF and Truncated SDF

The SDF at position $p \in \mathbb{R}^3$ is defined as:

$$\text{SDF}(p) = \begin{cases} \text{dist}(p, \partial \Omega) & \text{if } p \in \Omega, \\ -\text{dist}(p, \partial \Omega) & \text{otherwise}, \end{cases}$$

while the truncated SDF clamps the SDF as follows:

$$\text{TSDF}(p) = \begin{cases} -\tau & \text{if } \text{SDF}(p) \leq -\tau, \\ \tau & \text{if } \text{SDF}(p) \geq \tau, \\ \text{SDF}(p) & \text{otherwise}. \end{cases}$$

Depending on the available input data and the desired application, different formats may be more or less suitable. For example, colored images can be converted to NERF or NELF [15, 21], which optimizes the reconstruction error via differentiable rendering. Depth images can be fused into TSDFs as described in [3]; meshes can be converted into TSDFs, too, and also the inverse transformation is possible with variants of marching cubes [13]; point clouds can be turned into meshes or directly into TSDFs; etc.

2.2 Compression

Storing signed distance fields constitutes a computational and memory bottleneck, especially for time-varying data. In [23], a real-time compression method was proposed that uses a learnable, implicit temporal TSDF representation, combining independent color and texture encodings with learnable geometry compression. Multi-dimensional visual data compression based on the Tucker decomposition, combined with bit-plane coding, was applied to volumetric data in [1]. Later, [2] proposed compressing single-frame TSDFs with a block-based neural network. They showed that, although TSDF tensors usually cannot be written exactly as a low-rank expansion, in practice, one can store them in the low-rank TT representation with low compression error. Here, we build on the idea of tensor decompositions as a tool for compressing high-dimensional visual data. We address the task of compressing
Compressing TSDF with tensor methods not only allows one to work with data that otherwise would not fit the memory, but it can also be useful for:

1. Fusion, as demonstrated in [2];

2. Constant-time TSDF random access,\(^1\) which may be useful to test if a given point is inside or outside of a mesh, and potentially for ray-casting;

3. Constant-time computation of the TSDF gradient, used to compute surface normals [22];

4. Unlike neural network-based learning approaches, tensor methods rely on SVD-based schemes, which provide theoretical guarantees [17] and make the framework more interpretable.

Throughout this work, we use the term compression ratio (or just compression, if clear from the context) to denote the ratio between the number of coefficients used for the compressed format and the number of coefficients required to store the uncompressed tensor.

### 3 Tensor Formats

#### 3.1 Notation

**Tensor diagram notation.** To visualize tensor networks, we adopt Penrose’s graphical notation [18]. See Fig. 2 for visualization of a vector, a matrix and a matrix-vector product.

![Tensor diagram notation](image)

Figure 2: Each tensor is denoted as a node whose edges represent its dimensions. Whenever an edge is shared, tensor contraction is assumed: e.g., Fig. 2c depicts the matrix-vector product \( \sum_j M_{ij}v_j \).

#### 3.2 Tucker

The Tucker decomposition [24] factors a tensor of dimension \( D \) into a \( D \)-dimensional core tensor \( G \in \mathbb{R}^{r_1 \times \cdots \times r_D} \) and \( D \) matrices \( \{A_d\}_{d=1}^D, A_d \in \mathbb{R}^{r_d \times I_d} \). The format is defined as

\[
A[i_1, \ldots, i_D] = GA_1[; i_1] \ldots A_D[; i_D],
\]

where \( r_d \) are the Tucker-ranks. The Tucker decomposition has \( \mathcal{O}((\max_d[r_d])^D + \max_d[r_d] \cdot \max_d[I_d]) \) storage cost. See Fig. 3a for visualization of Tucker format in graphical notation.

\(^1\)The complexity of accessing a single element depends on the choice of tensor format and rank but is independent of the location of the query point within the tensor.
3.3 Tensor Train

The TT decomposition [17] factors a tensor of dimension $D$ into a sequence of $D$ 3-dimensional tensors. The format is defined as

$$A[i_1, \ldots, i_D] = Q_1[0, i_1, :]Q_2[:, i_2, :] \cdots Q_D[:, i_D, 0],$$

where the tensors $\{Q_d\}_{d=1}^D, Q_d \in \mathbb{R}^{r_{d-1} \times I_d \times r_d}$, are called TT-cores; and $r_d$ are the TT-ranks ($r_0 = r_D = 1$). The TT decomposition has $O(D \cdot (\max d[r_d])^2 \cdot \max d[I_d])$ storage cost, and leads to a linear tensor network; see Fig. 3b for a graphical example.

3.4 QTT

Quantics TT (QTT) is an extension of TT that includes reshaping\(^2\) the input into a tensor of shape $2 \times 2 \cdots 2 = 2^{\sum_{d=1}^D \log_2(I_d)}$, with $I_j$ the size of the $j$-th mode [10]. The sub-dimensions $x, y, z$ are then grouped side-by-side for each octet, which imitates the traversal of a $z$-space filling curve and makes QTT similar to a tensorized octree. Last, the resulting tensor is then subject to standard TT decomposition. This scheme is also connected to the wavelet transform [9].

Octet QTT. We also propose to reuse a variant of QTT with a base dimension of size eight instead of two by merging the three sub-dimensions of each octet into one. This format was originally proposed by Obukhov et al. [16]. In this way, the resulting tensor has shape $8 \times 2 \times \cdots \times 8 \times 2$. We refer to this format as OQTT for octet QTT and found it to reduce discontinuity artifacts (Section 5).

\(^2\)With zero-padding where needed
4 Proposed Method

We introduce T4DT, a method to compress high-resolution temporal TSDF fields with tensor decompositions discussed in Sections 3.3 - 3.4. We exploit that individual TSDF frames can be compressed into a low-rank TT decomposition with reasonable error [2]. Under the low-rank constraint, a zero-level set can be reconstructed with sufficiently good quality. However, this becomes more challenging when considering time-evolving data since uncompressed 4D grid scenes at fine spatial resolutions can range in the hundreds of GBs. Section 4 gives a high-level overview of our pipeline.

Algorithm 1 T4DT pipeline. Since the whole 4D scene is not expected to fit into memory, we first compress each frame individually. Next, the collection of individually compressed frames is assembled into a single compressed scene using a tree-like merge procedure $X'_0 \cup Y$. See Section 4.2 for the merge procedure details.

4.1 Framewise Decomposition

The first technical choice is the base decomposition at the frame level. We considered three variants: Tucker, TT, and QTT. In 3D, Tucker is an attractive choice since it is equivalent to TT plus an additional compression step along the second dimension. For the 4D case, we propose to combine the best of both worlds by using a TT-Tucker blend, where spatial dimensions are stored in Tucker formatted cores, and the temporal dimension is represented with a TT core. Each core or factor is responsible for a single dimension in both the TT and Tucker formats. However, the Tucker model shares the core tensor between all dimensions.

Many real-world datasets are sparse in the sense that most of the occupied volume is empty. An octree is an excellent choice to pack such sparse data into a compact representation. We argue that QTT is a tensor analog of an octree data structure. Indeed, it is easy to see that, padded and reshaped to $2^{\sum_{d=1}^{3} \log_2(I_d)}$, the scene becomes a piece-wise separated, reshaped set of octets after permutation of the corresponding sub-dimensions. Rank truncation is a way to reduce the number of coefficients to encode each octet sub-dimension.
4.2 Frame Concatenation

Since the original scene can by far exceed the available memory, we developed a progressive procedure to merge compressed frames into a single compressed scene. The algorithm is akin to the *pairwise summation* method to reduce round-off errors in numerical summation [5]: we stack the frames by pairs along the time dimension, recompress each pair via tensor rank truncation [17], and repeat the procedure recursively in a binary-tree fashion to ultimately yield a single 4D compressed tensor. See Fig. 4 for an illustration.

Note that this procedure is compatible with any format that supports rank truncation, which includes the TT and Tucker formats and mixtures thereof that allow for concatenation in compressed format. They are implemented via concatenation of the cores, increasing the rank of the resultant cores to a sum of the ranks of the operands. Afterward, rank truncation is applied to reduce the rank back to the desired maximal value with an SVD-like algorithm [17].

**QTT scene.** The scene stored in QTT format has the shape \( x_1 \times y_1 \times z_1 \times t_1 \times x_2 \times y_2 \times z_2 \times t_2 \times \cdots \times x_k \times y_k \times z_k \times t_k \), where \( x_i = y_i = z_i = t_i = 2 \) and \( \prod_{i=1}^{\log_2 I_1} x_i = W \), \( \prod_{i=1}^{\log_2 I_2} y_i = D \), \( \prod_{i=1}^{\log_2 I_3} z_i = H \), \( \prod_{i=1}^{\log_2 I_4} t_i = T \), with uncompressed scene shape \( W \times D \times H \times T \). In order to reconstruct the \( i \)-th frame from a scene compressed in this way, one must first compute the binary representation of \( i \) and then decompress the sub-tensor at \([i_0; \ldots; i_{\text{bit}_k}; \ldots; i_{\text{bit}_{k-1}}; \ldots; i_{\text{bit}_1}; \ldots; i_{\text{bit}_0}] \).
Results

5.1 Data

For our experiments, we use selected scenes from the CAPE dataset [14, 19]. The dataset consists of 3D meshes of 15 (10 male, 5 female) clothed people in motion. For each scene frame, we compute its TSDF and discretize it at resolution $512^3$, using the PySDF library. We use $\tau = 0.05$ in Eq. (2) to allow $\approx 10$ voxels with distinct levels near the TSDF zero level set. Since all scene frames must share a common coordinates frame, we recover the scene’s...
Algorithm 2 Conversion of a single 3D volumetric frame into OQTT format.

\[ I = I_1 \times I_2 \times I_3 \text{ - 3D grid} \]
\[ X \in \mathbb{R}^I \text{ - input 3D volumetric frame} \]
\[ R \text{ - maximal rank} \]
\[ Y \text{ - output compressed 3D volumetric frame} \]

Require:

\[ \text{for } d = 1, 2, 3 \text{ do} \]
\[ X \leftarrow \text{pad}(X, 2 \left[ \log_2 I_d \right]) \]
\[ \text{end for} \]
\[ X \leftarrow \text{reshape}(X, 2\Sigma_{d=1}^{3} \left[ \log_2 (I_d) \right]) \]
\[ X \leftarrow \text{permute}(X, (x_1, y_1, z_1, \cdots, x_\left[ \log_2 I_1 \right], y_\left[ \log_2 I_2 \right], z_\left[ \log_2 I_3 \right])) \]
\[ I_1 = \prod_{i=1}^{\left[ \log_2 I_1 \right]} x_i, \ I_2 = \prod_{i=1}^{\left[ \log_2 I_2 \right]} y_i, \ I_3 = \prod_{i=1}^{\left[ \log_2 I_3 \right]} z_i \]
\[ Y \leftarrow \text{TT}(X, R) \]

5.2 Influence of Tensor Ranks

We provide several error metrics computed for the TT-Tucker format, averaged across the first, middle, and last frames of the longshort-flying-eagle scene, for different spatial and temporal ranks in Fig. 5. See Appendix A.1 in supplementary material for the definition of the error metrics. Note that the time dimension is far from full-rank in the TT-Tucker format since the error metrics saturate well below the total sequence length of 284 frames.

The best compression/performance was obtained for the OQTT base format. Selected visualizations are provided in supplementary material in Appendix A.2: Fig. 9 shows error metrics for the TT format, as a function of the rank. Qualitative results are shown in Fig. 6 for the TT-Tucker format, in Fig. 7 for the TT format, and in Fig. 8 for QTT. For visual results of OQTT please refer to Fig. 1. In the cases of TT and TT-Tucker, severe rank truncation smooths the reconstructed surface and erodes smaller features like fingers or facial details. In contrast, QTT preserves more details at the cost of discontinuities due to the separation and independent compression of octet sub-dimensions. OQTT reduces these discontinuity artifacts by encoding the octets as a single dimension without compressing the ranks between sub-dimensions of a single octet. See Fig. 1 for qualitative results for OQTT. We also present quantitative OQTT compression results in Table 1, at a rank that offers a good compromise between visual quality and compression rate.

6 Failure Cases and Limitations

We did not observe significant failures while working with the CAPE and AMA datasets. However, how T4DT fares for 3D data that are not structurally sparse (i.e., not mostly free space) remains to be tested. Unfortunately, there are no such scenes in the two datasets.
We do note that TT decomposition is not invariant against the rotation of the input. Hence, one could attack the scheme by rotating the scene w.r.t. the grid axes in a way that maximizes the reconstruction error. A straightforward measure to mitigate the influence of the grid orientation is to pre-rotate the 3D scene to its principal axes before grid sampling and OQTT compression.

7 Conclusions

We have presented T4DT, a scalable and interpretable compression pipeline for 3D time sequence data based on tensor decomposition. Our scheme improves over the related TT-TSDF method in two ways: (i) We are able to process temporally varying data. We can do so even though the TSDF field takes hundreds of GBs, and our method works fully in-memory; (ii) We tested various tensorization schemes, such as a TT-Tucker hybrid, the QTT format, and the new OQTT variant, and found OQTT to outperform previous decompositions. OQTT is tailored specifically to 3D volumetric scene representations, and our experiments quantitatively and qualitatively support its special reshaping of the 3D grid.
References


