TT-NF: Tensor Train Neural Fields

Anton Obukhov, Mikhail Usvyatsov, Christos Sakaridis, Konrad Schindler, Senior Member, IEEE, and Luc Van Gool

Abstract—Learning neural fields has been an active topic in deep learning research, focusing, among other issues, on finding more compact and easy-to-fit representations. In this paper, we introduce a novel low-rank representation termed Tensor Train Neural Fields (TT-NF) for learning neural fields on dense regular grids and efficient methods for sampling from them. Our representation is a TT parameterization of the neural field, trained with backpropagation to minimize a non-convex objective. We analyze the effect of low-rank compression on the downstream task quality metrics in two settings. First, we demonstrate the efficiency of our method in a sandbox task of tensor denoising, which admits comparison with SVD-based schemes designed to minimize reconstruction error. Furthermore, we apply the proposed approach to Neural Radiance Fields, where the low-rank structure of the field corresponding to the best quality can be discovered only through learning.

Index Terms—Deep learning, neural fields, radiance fields, sampling, tensor decompositions, tensor train, voxels.

I. INTRODUCTION

Following the growing interest in deep neural networks, learning neural fields has become a promising research direction in areas concerned with structured representations. However, precision is usually at odds with the computational complexity of these representations, which makes training them and sampling from them a challenge. In this paper, we investigate interpretable low-rank neural fields defined on dense regular grids and efficient methods for learning them. Since, in extreme cases, the dimensionality of such fields can exceed the memory size of a typical computer by several orders of magnitude, we look at the problem of learning such fields from the angle of stochastic methods.

Tensor decompositions have become a ubiquitous tool for dealing with structured sparsity of intractable volumes of data. We focus on the Tensor Train (TT) [1], also known as the Matrix Product State in physics. TT is notable for its high-capacity representation, efficient algebraic operations in the low-rank space, and support of SVD-based algorithms for data approximation. As such, we consider TT-SVD [1] and TT-cross [2] methods for obtaining a low-rank representation of the full tensor. While TT-SVD requires access to the full tensor at once (which might already be problematic in specific scenarios), TT-cross requires access to data through a black-box function, computing (or looking up) elements by their coordinates on demand. Both methods operate under the assumption of noise-free data and are not guaranteed to output sufficiently good approximations in the presence of noise.

While noise in observations is challenging for SVD-based schemes and requires devising tailored approaches to different noise types and magnitude [3], exploiting the low-rank structure of the field driven by data is even more challenging [4], [5] and typically resorts to the paradigm of data updates through algebraic operations on TT.

In this work, we take a step back and leverage the modern deep learning paradigm to parameterize neural fields as TT, coined TT-NF. Through deep learning tooling with support for automatic differentiation and our novel sampling methods, we obtain mini-batches of samples from the parameterized neural field and perform optimization of a non-convex objective defined by a downstream task. The optimization comprises the computation of parameter gradients with backpropagation and...
parameter updates with any suitable technique, such as SGD. We analyze TT-NF and several sampling techniques on a range of problem sizes and provide reference charts for choosing a sampling method based on memory and computational constraints. Next, we define a synthetic task of low-rank tensor denoising and demonstrate the superiority of the proposed optimization scheme over several SVD-based schemes. Finally, we consider the formulation of Neural Radiance Fields (NeRF) [6], and propose a simple modification to TT-NF, termed QTT-NF, for dealing with hierarchical spaces.

Our contributions in this paper:
1) TT-NF – compressed low-rank neural field representation defined on a dense grid;
2) QTT-NF – a modification of TT-NF for representing hierarchical spaces, such as 3D voxel grids seen in neural rendering (see Fig. 1);
3) Efficient algorithms for sampling from (Q)TT-NF and learning it from samples, designed for deep learning training.

The rest of the paper is organized as follows: Section II discusses the related work; Section III introduces notations from the relevant domains; Section IV presents the proposed contributions; Section V demonstrates the practical use of the proposed methods; Section VI discusses implementation aspects and limitations of the proposed method; Section VII concludes the paper.

II. RELATED WORK

1) Tensor decompositions: Higher-order tensor decompositions have been found helpful for several data-based problems [7]. Oseledec’s [11] introduced the Tensor Train (TT) decomposition, which offers a compressed low-rank tensor approximation that is stable and fast. The TT decomposition has also been used to approximate tensors with linear complexity in their dimensionality via the TT-cross approximation [2]. With the rise of deep learning, tensor-based methods have been integrated into neural networks, e.g., gradient selection with TT-cross approximation [8]. We review tensor-based methods for network compression in the next paragraph and refer the reader to [9] for a detailed overview of similar works. On the software side, along with general deep learning frameworks [10], [11], several tensor-centric frameworks have emerged [12], [13], [14].

2) Neural network compression with tensors: Low-rank bases were utilized [15] to approximate convolutional filters and drastically speed up inference via separating filter depth from spatial dimensions. Differently, low-rank decomposition was applied on all 4 dimensions of the standard convolutional kernel tensors [16]. Subsequent works employed more general tensor decompositions, notably the TT decomposition, to massively compress fully connected layers [17] or both fully connected and convolutional layers [18], with minor accuracy losses. Other decompositions, such as tensor rings, were also explored in the same context [19]. Higher-order tensor factorization was applied to the entire network instead of separately to individual layers [20]. In a similar vein, several works proposed to learn the basis and coefficients of each layer, thus enabling disentangled compression and multitask learning [21], [22], [23]. While most of the aforementioned methods examine general convolutional networks, we focus specifically on compressing neural fields.

3) Tensor decompositions in 3D representations: Compact representations and sampling techniques for compressed volume rendering are reviewed in [24]. Multiple tensor approximation models were analyzed regarding volume visualization [25]. More recently, compression of 3D volumes over regular grids was addressed with a general higher-order singular value decomposition [26]. The TT decomposition has also been used [5] to compress 3D scenes represented by volumetric distance functions. We review neural-field-based methods separately in the next paragraph.

4) Neural fields: DeepSDF [27] was among the first practical examples of using neural fields to represent 3D shapes as signed distance functions. A view of the 3D representation at training and inference would be obtained with differentiable ray casting or Marching Cubes [28]. These ideas are further developed in MVSD [29], designed for the Multi-View Stereo setting. The usual volumetric type of representation is replaced by a surface-based one in [30], who learned bidirectional reflectance distribution functions that enable novel view synthesis for unconstrained real-world scenes. For a comprehensive overview of neural fields in visual computing, we refer the reader to [31].

5) Neural radiance fields: The seminal paper [6] introduced the concept of a NeRF as an end-to-end implicit differentiable function that maps viewpoints to scene renderings and showed that it could be effectively optimized. The aliasing artifacts of NeRF caused by single-ray sampling were addressed via a continuously-valued frustum-based rendering approach [32]. Another work [33] extended NeRF to jointly model multiple objects of the same class and decouple the object’s content from its pose. The area advances quickly; we further discuss works focusing on representation sparsity and efficiency. Plenoxels [34] employed a sparse 3D-grid representation based on spherical harmonics and regularization for novel view synthesis, alleviating the need for neural components. InstantNGP [35] improved the training efficiency by using a hash encoding for the input feature vectors, allowing for a much smaller network. NeRF architecture was reformulated and stored the model as a sparse voxel grid to enable real-time rendering via precomputation [36]. Faster convergence was achieved using a post-activation interpolation and guiding the optimization via prior knowledge of the problem [37]. TensoRF [38] implemented low-rank constraints in 3D on the voxel grid parameterization and achieved impressive results in the NeRF setting. Further, HexPlane [39] and K-planes [40] lifted this idea into yet another dimension to model temporal sequences of moving objects in 3D.

III. NOTATION

1) Tensor diagram notation: Diagrams are an efficient visualization tool for interactions between tensors. A tensor is drawn as a node with the number of legs matching its number of dimensions. For example, a matrix $W \in \mathbb{R}^{m \times n}$ is drawn as $\otimes$ and a vector $x \in \mathbb{R}^n$ looks as $\bullet$. Their product $Wx$ looks as a connection along the dimension being eliminated as a result of the operation: $\otimes \bullet \otimes$. A diagram of nodes with their connections reflects what is called a “tensor network”.

2) Tensor contraction: computes a product of the entire tensor network. The result of this operation is a single tensor with dimensions corresponding to free legs inside the tensor network.

3) Tensor train decomposition: The TT format represents a D-dimensional array (tensor) $A \in \mathbb{R}^{M_1 \times M_2 \times \cdots \times M_D}$ with modes $M_i$ as a product of $D$ three-dimensional core tensors $C^{(i)} \in \mathbb{R}^{R_{i-1} \times M_i \times R_i}$, called TT-cores (see Fig. 2). The tuple $(R_0, R_1, \ldots, R_D)$ is called a TT-rank of the decomposition; it defines the degree of approximation of $A$. By convention $[1]$, $R_0 = R_D = 1$, $R_{\max} = \max(R_0, \ldots, R_D)$ is also
called the rank of the decomposition. An element of $A$ at indices $(i_1, \ldots, i_D)$ can be computed as follows:

$$A_{i_1, \ldots, i_D} = \sum_{\beta_1, \ldots, \beta_D} C_{\beta_1, \ldots, \beta_D}^{(1)} \cdot C_{\beta_1, \ldots, \beta_D}^{(2)} \cdot \ldots \cdot C_{\beta_1, \ldots, \beta_D}^{(D)}$$

(1)

Block TT [41] attaches a “block” dimension in place of the last rank $R_D$. The difference between the two is subtle, as both formats can be converted to each other. However, Block TT is more suitable for describing multi-valued neural fields. We thus assign a special meaning to $R_D$—it will signify the “payload” dimension of our neural field. We will omit the word “block” in the remaining text and always assume Block TT. Refer to Fig. 2 for the diagram.

4) **QTT**: Mode Quantization refers to the introduction of artificial dimensions into the represented tensor. For example, a 3D tensor of function values on the lattice $16 \times 16 \times 16$ can be represented as a tensor of shape $(2_1 \times 2_2 \times 2_3 \times 2_4) \times (2_1 \times 2_2 \times 2_3 \times 2_4) \times (2_1 \times 2_2 \times 2_3 \times 2_4)$. Here, color-coded factors denote one of the axes $X$, $Y$, and $Z$ they explain; subscripts denote the artificially-introduced levels of hierarchy, $\times$ delimits represented dimensions, $\cdot$ denotes merged (flattened) dimensions and parentheses are for visual convenience. This 12-dimensional cube with side 2 can then be represented using TT as in Fig. 2. However, when the number of introduced levels of hierarchy is the same between modes of the original tensor (four in the given example), it is often beneficial [42] to introduce low-rank structure between levels of hierarchy, rather than individual factors. The resulting low-rank representation describes a tensor of shape $8 \times 8 \times 8 \times 8$ with the following factorization pattern called QTT: $(2_1 \cdot 2_1 \cdot 2_1) \times (2_2 \cdot 2_2 \cdot 2_2) \times (2_3 \cdot 2_3 \cdot 2_3) \times (2_4 \cdot 2_4 \cdot 2_4)$. Notably, such permutation of dimension factors corresponds to 3D space traversal using Morton code [43], also known as Z-order (see Fig. 1). Connections with octrees used in rendering can also be drawn.

We define our neural field on a 3D voxel grid as shown in Fig. 3. Dimensions of the voxel grid are chosen equal to 2D, resulting in a hierarchy of $D$ levels. Together with the Block structure discussed above and QTT, we arrive at the proposed representation, which has $D$ cores with all modes equal to eight, and the payload dimension corresponding to the number of values to store in each voxel.

5) **Tensor decomposition**: The term may refer either to the decomposition scheme (e.g., as in the figures above) or the process of obtaining values of the decomposition (e.g., TT-cores $C$) from elements of the full tensor $A$. While SVDbased schemes employ the latter meaning, we focus on the former by parameterizing decomposition given its configuration, which includes modes and TT-rank.

![Fig. 2. Tensor diagram of the (block) tensor train decomposition [41]. The low-rank tensor of shape $M_1 \times \ldots \times M_D \times R_D$ is represented as a product of $D$ TT-cores $C^{(i)}$, each being a tensor of shape $R_{i-1} \times M_i \times R_i$. The TT-rank $(1, R_1, \ldots, R_D)$ defines the degree of approximation. The case of $R_D = 1$ corresponds to TT decomposition [1].](image1)

![Fig. 3. Tensor diagram of a quantized tensor train decomposition [44] of a 3D voxel grid of shape $X \times Y \times Z$ and $R_D$ values in each voxel. All three dimensions admit factorization into $D$ levels of hierarchy (e.g., $X = X_1 \times X_2 \times \cdot \cdot \cdot \times X_D$). We group factors by levels into tuples $(X_i, Y_i, Z_i)$, and introduce the low-rank bonds between the levels of the hierarchy.](image2)

### Table I

| Method         | Observation access pattern | Noise in observations |
|----------------|---------------------------|-----------------------|
| TT-SVD         | Full tensor               | Not supported         |
| TT-Cross       | On-demand, pattern defined by dimensions and TT-rank | Not supported |
| TT-OI          | Full tensor               | Sub-gaussian          |
| TT-NF (our)    | On-demand, flexible batch size and access pattern | Any supported by the choice of the loss function |

6) **Sampling**: refers to obtaining elements of the full tensor from its decomposition at a given list of indices. This operation could be done in several ways, the simplest being Tensor Contraction followed by subsampling at the required indices. When the contraction is undesired or intractable, an alternative way is evaluating full tensor elements through the decomposition equation, such as (1). While mathematically straightforward, the subtleties of the chosen sampling algorithm result in a large variance in efficiency when used within the optimization loop due to the need for automatic differentiation, as we discuss in Section IV-B.

### IV. Method

We introduce TT-NF as a parameterization of the TT decomposition discussed in Section III and use it within a deep learning framework with automatic differentiation support. This paradigm change differs from previous methods for obtaining tensor decompositions, relying on matrix decompositions (SVD, QR, etc.) and algebraic operations in the TT format. Each such scheme (collectively called “SVD-based”) comes with its own set of limitations, summarized in Table I. TT-SVD [1] assumes access to all elements of the full tensor $A$ in memory, which may be intractable in specific large-scale scenarios and does not support noise in observations. TT-Cross [2] accepts a black-box function for computing elements of the full tensor on-demand, which suits large-scale problems, but lacks the flexibility in choosing the number of samples through which update is performed (batch size); this number is defined purely by the configuration of TT decomposition (dimensions and TT-rank). Likewise, it does not support noise in observations. TT-OI [3]
improves upon TT-SVD and supports zero-mean independent sub-gaussian noise in observations but inherits scalability issues. Our method stands out due to its flexibility of optimization parameters choice (e.g., batch size) and resilience to various types of noise in observations, controlled through the choice of the loss function.

A. Initialization of TT-NF

Given the field’s dimensions, we first choose its TT-rank. For that, we choose the value of $R_{\text{max}}$ and set TT-rank to the maximum possible values according to [1], not exceeding $R_{\text{max}}$.

Following the best practices in the deep learning literature, we initialize parameters of TT-NF from scratch using the normal distribution with scale $\hat{\sigma}$ computed such that the full tensor elements computed using (1) have a pre-defined scale $\sigma$, as shown in (2):

$$\hat{\sigma} = \exp \left( \frac{1}{2D} \left( 2 \log \sigma - \sum_{i=1}^{D} \log R_i \right) \right).$$

Alternatively, if the full tensor is stored in memory, TT-core parameters can be initialized using the output of any of the SVD-based schemes, leading to faster convergence.

Algorithm 1: Memory-Efficient Sampling from TT-NF for Deep Learning Frameworks. Automatic differentiation paths are highlighted in blue. Refer to Section IV-B for More Details.

Require:

- $D$ - number of tensor dimensions,
- $B$ - number of samples,
- $(1, R_1, \ldots, R_D)$ - TT-rank,
- $(M_1, \ldots, M_D)$ - TT-modes,
- $(C^{(1)}, \ldots, C^{(D)})$ - TT-cores representing $A$,
- $((i^{(1)}_1, \ldots, i^{(1)}_D), \ldots, (i^{(B)}_1, \ldots, i^{(B)}_D))$ - indices.

Ensure:

- $v = (A_{(i^{(1)}_1, \ldots, i^{(1)}_D)}, \ldots, A_{(i^{(B)}_1, \ldots, i^{(B)}_D)})$ - samples without computing the whole $A$.

1: $\pi \leftarrow (1, \ldots, B)$ $\triangleright$ forward permutation
2: $\sigma \leftarrow (1, \ldots, B)$ $\triangleright$ inverse permutation
3: $v \leftarrow C^{(1)}_{1, \pi_1}$ $\triangleright B \times R_1$
4: for $k \leftarrow 2$ to $D$
  5: $i_k \leftarrow \pi(i_k)$
  6: $v, \pi_k \leftarrow \text{BIMVP}(C^{(k)}, i_k, v)$ $\triangleright B \times R_k$
  7: $\sigma_k \leftarrow \pi_k^{-1}$ $\triangleright$ invert $k^{th}$ permutation
  8: $\pi \leftarrow \pi_k(\pi)$ $\triangleright$ update forward perm.
  9: $\sigma \leftarrow \sigma(\pi_k)$ $\triangleright$ update inverse perm.
10: end for
11: $v \leftarrow \sigma(v)$ $\triangleright$ recover samples order
12: return $v$ $\triangleright B \times R_D$

B. Sampling From TT-NF

Obtaining samples from TT-NF is mathematically straightforward using (1). In deep learning frameworks, one way to obtain a batch of $B$ samples at indices $((i^{(1)}_1, \ldots, i^{(1)}_D), \ldots, (i^{(B)}_1, \ldots, i^{(B)}_D))$ amounts to using $\text{index}_\text{select}$ operation on each TT-core $C_i$ along mode $M_i$ to obtain batches of core slices (matrices) of shapes $B \times R_{i-1} \times R_i$, and then applying $\prod_{i=2}^{D} (B)$ (batched matrix multiplication) operation to them. This sampling method (aliased $v1$) leads to space complexity of $O(BDR_{\text{max}}^S)$ and quickly becomes unusable as slices of parameterization are replicated in memory for each sample. Moreover, the scaling issue worsens as we require keeping all intermediate computations after each $\text{perm}$ operation and allocating memory for gradients to enable automatic differentiation.

The prevention of model parameter replication enabled the scaling of modern neural networks with millions of parameters, which are trained using minibatches of thousands of samples. The cornerstone of efficient scaling is a set of specialized layers (e.g., Linear), which accept a batch of inputs and compute mappings using only one instance of parameters. During the backward pass, gradients w.r.t. parameters are accumulated from samples with predictable, constant space scaling.

With that in mind, we design our efficient sampling method (aliased $v2$) by leveraging Linear layer functionality. Given a batch of indices as above, we start by taking $(i^{(1)}_1, \ldots, i^{(1)}_D)$ and produce a batch of intermediate vectors $v$ of shape $B \times R_1$ (the dimension corresponding to $R_0 = 1$ is ignored). For each subsequent TT-core $C_i, i = 2, D$, we split the inputs $v$ according to which $M_i$ slices of $C_i$ will be used to perform vector-matrix multiplication of each sample. Because Linear layers require samples in the minibatch to be packed densely, we perform a permutation $\pi$ of $v$ to align samples in the minibatch before multiplying them with the respective weight matrices. We additionally maintain the inverse permutation $\sigma$ to restore the order of samples after processing $v$ with the last TT-core. The output of each step has the shape $B \times R_{i+1}$ compatible with the input to the next step until reaching the last step, where it is of the shape $B \times R_D$. Algorithm 2 outlines the details of the algorithm.

The resulting space complexity of $v2$ sampling is reduced to $O(BDR_{\text{max}}^S)$, with the absorbed scaling factor just $2 \times$ that of the $v1$ method due to permutations. The memory footprint of $v2$ is roughly $R_{\text{max}}$ times smaller than $v1$, which enables practical use for TT-NF optimization as the rank increases.
We compare three sampling schemes discussed in Section IV, as well as the traditional tensor contraction scheme. As seen in the plots, \( \mathbf{v}_2 \) (Section IV-B) scheme requires orders of magnitude fewer floating point operations (FLOPs) and memory than the contraction scheme and outperforms naive \( \mathbf{v}_1 \) sampling in memory requirements. Additionally, \( \mathbf{v}_3 \) (Section IV-C) offers extra speedup on top of \( \mathbf{v}_2 \) without loss of representation capacity, bringing its parameterization closer to the theoretical number of degrees of freedom of the tensor train manifold. Lower is better. Best viewed in color.

Fig. 4 provides a reference chart for choosing an optimal sampling scheme based on an uncompressed field size of \( 2^{30} \), batch size, \( R_{\max} \)-space, and time constraints. Refer to additional charts for different field and batch sizes in Fig. 5. As can be seen, \( \mathbf{v}_2 \) consistently outperforms \( \mathbf{v}_1 \) in memory requirements and sampling through tensor contraction followed by indexing in both memory and FLOPs, given the batch sizes specified in the plot. Additionally, we introduce a reduced parameterization and an associated \( \mathbf{v}_3 \) sampling method in Section IV-C.

C. Ranks and Parameterizations

1) TT-rank selection: We clarify the procedure of choosing a TT-rank, mentioned in Section IV-A. To parameterize TT-NF, we need to know the dimensions (modes) of the tensor \( M_i, i = 1, D \), the payload dimension \( R_D \), and a scalar hyperparameter \( r \), defining the degree of compression. A TT-rank resulting from a TT-SVD [1] algorithm is bounded as in [45, Eq. 20], with the only difference in added \( R_D \) term to account for the discussed block structure:

\[
1 \leq R_k \leq R_{\max}^k, \text{ where } k = 1, D-1, \text{ and }
R_{\max}^k \equiv \min \left( \prod_{i=1}^{k} M_i, R_D \prod_{i=k+1}^{D} M_i \right). \tag{3}
\]

This means that the maximal TT-rank forms a “pyramid” of integers, where each position \( i \) is assigned a minimum of products of modes to the left and right of \( i \). We denote \( R_{\max} \) as the maximum value of TT-rank and \( t_{\max} \) as the position of the maximum value (the “peak”). As shown in Figs. 4 and 5, \( R_{\max} \) can go as high as 1024 in \( 2^{20} \) and 32768 in \( 2^{28} \) sizes. To achieve compression with our parameterizations, we clamp the “pyramid” at a chosen value of rank \( r \in [1, R_{\max}] \), thus making the TT-rank “trapezoid”. This range corresponds to the X-axes of the aforementioned figures.

2) Full TT-NF parameterization: Full parameterization assumes the allocation of learned parameters to each element of TT-cores \( C^{(i)}, i = 1, D \). Such parameterization is compatible with \( \mathbf{v}_1, \mathbf{v}_2 \) sampling schemes, as well as tensor contraction and subsampling.

3) Reduced TT-NF parameterization: A TT manifold of a fixed TT-rank has a certain number of Degrees of Freedom (DOF) [45], which is smaller than the number of learned parameters of TT-NF with \( \mathbf{v}_2 \) sampling. Practical mappings from DOF to learned parameters exist but require complex transformations [46]. The latter work provides insights into a simpler form of reduced parameterization. This becomes possible once the rank becomes large enough that some TT-cores have square “matricizations”, meaning that either \( R_{i-1} M_i = R_i \) or \( R_{i+1} = M_i R_i \). Such cores can be replaced with fixed identity matrices (reshaped into the original core shapes), leading to parameter count reduction. Additionally, knowledge of such reduced parameterization suggests a modification to the \( \mathbf{v}_2 \) sampling algorithm, aliased \( \mathbf{v}_3 \).

Given \( r \), we denote \( 1 \leq p \leq q \leq D \) indices of the first and last clamped values in TT-rank. It is easy to verify that TT-cores \( C^{(i)} \) to the left of \( p \) and to the right of \( q \) should have square matricizations. Indeed, a TT-core at position \( i < p \) has a shape \( R_{i-1} \times M_i \times R_i = R_{i-1}^\times \times M_i \times R_i \), thus its left matricization (merging the first two dimensions, denoted \( \text{leftmat} \)) is square.

As shown in [46], such cores can effectively be replaced with fixed identity matrices (reshaped into the original core shapes) without any loss of representation power of such TT-NF. The sparsity of identity matrices suggests the possibility of skipping steps of the \( \mathbf{v}_2 \) sampling algorithm and changing several matrix
Fig. 5. Space-time complexity of sampling from TT-NF of sizes $2^{20}$ (left) $2^{30}$ (right) with various methods, batch sizes, and ranks. We compare three sampling schemes discussed in Sections IV and IV-C, and the traditional tensor contraction scheme. The optimal choice of sampling scheme depends on the rank, problem, and batch sizes. Lower is better. Best viewed in color.

multiplication operations (\texttt{Linear} layers) to a single indexing operation.

Considering the left side of such a TT-NF again, if $C^{(1)}_{\ell_1} = I_{R_1}$, then on the first step of Algorithm 1 $\mathbf{v} = e_{i_1}$, where $e_j$ denotes an ort with 1 in $j$-th position. By induction, if $\mathbf{v} = e_{i_{\text{left}}}$ and $\text{left\text{mat}}(C^{(k)}) = I_{R_k}$, then $C^{(k)}_{\ell_k}:i_k,v = e_{i_{\text{left}}}M_k+i_k$. Such index propagation rule allows us to skip BIMVP computation for all such cores and index $v$ directly along the first rank mode of the first parameterized TT-core at position $p$. The index propagation rule for the right side of TT-NF can be derived similarly. The full algorithm for $v_3$ sampling can be found in Algorithm 3.

Notably, in the extreme case of decomposition rank $r = R_{\text{max}}$, only a single TT-core contains learned parameters, corresponding to the uncompressed tensor. In this case, sampling with $v_3$ becomes the indexing operation along all three modes of the TT-core, leading to zero FLOPs. This corner case explains the drop of FLOPs line around the far end of the rank axis. The remaining saw-like drops occur when $r$ increases above intermediate values, making one or two more TT-cores become reshaped identity matrices. While the absolute gain in FLOPs is not substantial when rank $r$ is not large, we observe a noticeable speed-up of $24 \times$ with $v_3$, compared to training with $v_2$, due to fewer sequential invocations of \texttt{Linear} layer functions.

The reduced parameterization is compatible with $v_1$, $v_2$, $v_3$ sampling schemes, as well as tensor contraction and subsampling.

4) Conversion from full to reduced parameterization: Given a Full parameterization, conversion to a Reduced one can be performed in a single pass over TT-cores. Indeed, starting from the first TT-core and repeating until reaching position $p$, we can compute the left matricization of the current TT-core and absorb (through matrix multiplication) this square matrix into the right-hand-side TT-core. The current core is then replaced with an identity matrix and reshaped into the original shape of the core. A similar process starting from the last TT-core until reaching position $q$ completes the conversion.

D. Neural Rendering

The core of neural rendering is comprised of two main components: (1) a neural field and (2) differentiable ray marching.

1) Neural field: In the NeRF paper [6], the neural field is represented with a neural network mapping $f_\theta : (x, d) \rightarrow (c, \sigma)$, where $x$ is a position in some scene-centric frame of reference, $d$ is a viewing direction pointing at a camera, $c$ is a view-dependent color (e.g., 8-bit RGB channels), and $\sigma$ is a view-independent volume density at $x$.

Other works [34] popularized the usage of spherical harmonics to represent view-dependent color variations. Effectively, the viewing direction $d$ is removed from the parameterization; instead, color descriptors of sufficient size are regressed at each location along with density $\sigma$. Such color descriptors can be coefficients in a pre-defined spherical harmonics basis, effectively parameterizing color as a function on a sphere. Alternatively, the transformation of the learned descriptor and viewing direction can be learned with a dedicated “shading” MLP, as done in TensorRF [38]. Empirically, degree two harmonics appear sufficient...
Algorithm 3: Memory-Efficient Sampling From a Reduced Parameterization of TT-NF (v3). Auto-differentiation Paths are Highlighted in Blue. Refer to Section IV-C for More Details.

```
Require:
    D - number of tensor dimensions,
    B - number of samples,
    1 \leq p \leq q \leq D - parameterized cores range,
    (I_1, R_1, \ldots, R_D) - TT-rank,
    (M_1, \ldots, M_D) - TT-modes,
    (C^{(p)}, \ldots, C^{(q)}) - TT-cores representing A,
    ((i_1^{(1)}, \ldots, i_D^{(1)}), \ldots, (i_1^{(B)}, \ldots, i_D^{(B)})) - indices.
Ensure:
    v = (A_{i_1^{(1)}, \ldots, i_D^{(1)}}, \ldots, A_{i_1^{(B)}, \ldots, i_D^{(B)}}) - samples without computing the whole A.

1:   i_{\text{left}} \leftarrow (0, \ldots, 0) \quad \triangleright B \text{ left indices}
2:   i_{\text{right}} \leftarrow (0, \ldots, 0) \quad \triangleright B \text{ right indices}
3:   for k \leftarrow 1 to p - 1 do
4:       \triangleright Propagate left indices
5:       i_{\text{left}} \leftarrow i_{\text{left}} + M_k + (i_k - 1)
6:   end for
7:   for k \leftarrow q + 1 to D do
8:       \triangleright Propagate right indices
9:       i_{\text{right}} \leftarrow i_{\text{right}} + (i_k - 1) \times R_k
10:  end for
11:  if p = q then
12:     \triangleright Directly index v in C^{(p)}
13:   else
14:     v \leftarrow C^{(p)}_{i_{\text{left}}, i_{\text{right}}, i_{\text{right}} + R_D}
15:   end if
16:  v \leftarrow C^{(p)}_{\text{left}, c, \text{right}} + R_D

17:  \pi \leftarrow (1, \ldots, B) \quad \triangleright forward permutation
18:  \sigma \leftarrow (1, \ldots, B) \quad \triangleright inverse permutation
19:  v \leftarrow C^{(p)}_{\text{left}, \sigma c, \text{right}} \quad \triangleright B \times R_p
20:  for k \leftarrow p + 1 to q do
21:      i_k \leftarrow \pi_{i_k} \quad \triangleright align mode indices
22:      v, \pi_k \leftarrow \text{BIMVP}(C^{(k)}, i_k, v) \quad \triangleright B \times R_k
23:      \sigma_k \leftarrow \pi^{-1}_k \quad \triangleright invert k-th permutation
24:      v \leftarrow \pi v \quad \triangleright update forward perm.
25:      \sigma \leftarrow \sigma \sigma_k \quad \triangleright update inverse perm.
26:  end for
27:  v \leftarrow \sigma v \quad \triangleright reorder order, B \times R_d
28:  v \leftarrow v_{\text{right}}, i_{\text{right}} + R_D \quad \triangleright B \times R_d
29:  end if
30:  return v \quad \triangleright B \times R_d
```
uniformly to probe the field between hardcoded near and far planes, and then 128 more samples are obtained to increase sampling density at places with content. Voxel grid neural fields [34] alter the sampling procedure: rays are filtered based on whether they intersect with the voxel grid, and points inside the grid are sampled uniformly between near and far intersection points. A floating-point coordinate sampling of voxel grids is resolved through trilinear interpolation of payload vectors. We follow the same procedure in our experiments with QT-T-NF.

Samples (c_i, \sigma_i) at positions r_i of a ray r are accumulated using the following equations:

\[ C(r) = \sum_{i=1}^{N} T_i (1 - \exp(-\sigma_i \delta_i)) c_i, \]

Here \( \delta_i = \|r_i - r_{i-1}\|_2 \), \( T_i \) represents optical transmittance of the ray at position \( r_i \), and the last term denotes contribution of the position to the accumulated color.

V. EXPERIMENTS

A. Tensor Denoising

To compare TT-NF with SVD-based schemes, we use a synthetic task of tensor denoising: given a noisy observation \( \mathcal{Y} \in \mathbb{R}^{M_1 \times \ldots \times M_D} \) of a tensor \( \mathcal{X} \) with a known TT structure as in (1), the task is to compute \( \hat{\mathcal{X}} = \text{argmin} \| \mathcal{X} - \mathcal{A} \|_F^2 \). For the experiments, we first choose tensor modes and TT-rank with \( R_d = 1 \), generate \( \mathcal{X} \) as in the TT-NF initialization scheme from Section IV-A with \( \sigma = 1.0 \), and perform tensor contraction. To simulate noisy observations \( \mathcal{Y} \), we sample noise \( \mathcal{Z} \) from independent zero-mean Normal or Laplace distributions with a chosen scale and compute \( \mathcal{Y} = \mathcal{X} + \mathcal{Z} \).

TT-SVD [1] is a deterministic¹ algorithm; it does not take any hyperparameters and produces a TT decomposition in a single pass over data. TT-OI [3] performs several passes. TT-cross has several stopping settings, as it runs an iterative maximum volume algorithm [47] on each iteration of the main algorithm. We take the defaults provided by the tntorch package [13]. On the side of the non-convex optimization family, we minimize a loss function between samples from the observation and TT-NF. We choose a loss function best suited for the type of observation noise: L1 for Laplace and L2 for Normal. Contraction denotes the usage of all possible elements in the loss, whereas Sampling considers only mini-batches of samples. Fig. 6 demonstrates the performance of all methods on the problem of size 2^20 under varying noise types, scale, and TT-NF rank. Optimization lasts 1000 steps, with a batch size of 4096 elements, Adam optimizer [48] with default settings, learning rate warming up during the first 5% steps to 3e-2 and decaying exponentially to 3e-4. We ran all experiments 10 times and reported plots with 1-std error bars. All runs are completed on a single GPU.

TT-SVD provides a reasonable baseline, which we use as initialization for TT-NF. TT-OI works better than TT-SVD on a subset of Normal noise levels and ranks. TT-cross works only in a noise-free setting. Contraction, which can be considered

¹To the extent we can call SVD deterministic.
Fig. 6. The resilience of various tensor regression methods of size $2^{20}$ to additive noise. We consider zero-mean normal and laplace noise with varying scales added to random ground truth with unit variance and known TT decomposition. We report root mean squared errors (RMSE) of tensors regressed by each method. Our sampling methods (v2 and v3) outperform other methods in noisy settings, including those specifically designed to work with noisy observations (TT-OI [3]) and those computing a fraction of elements at each optimization step (TT-cross [2]). See discussion in Section V-A. Lower is better. Best viewed in color.

Fig. 7. Sweeps of TT-rank, number of training steps, and spherical harmonics components per channel when training QTT-NF on the “Lego” scene. The remaining parameters are fixed to the baseline values (represented with a large dot in plots); See Section V-B. Higher is better.

as full-batch gradient descent, does not improve much upon TT-SVD. We conjecture that it gets stuck in saddle points, a recurring argument seen in discussions of the advantages of SGD over GD in the deep learning literature. As can be seen, Sampling methods outperform all others in settings with the presence of noise.

B. Neural Radiance Fields

Following the fast-pacing domain, we test a QTT-NF variant of our representation (Fig. 3) in the neural radiance fields (NeRF) setting. The task of learning voxelized radiance fields from data assumes access to views of a single scene with known camera poses in some frame of reference. The objective is to regress features of the voxel grid such that when passed through differentiable shading and ray marching algorithms, the projected images would correspond to the ground-truth data, and views from the held-out set would exhibit high PSNR. Such a data-driven problem formulation does not permit the usage of existing SVD-based solutions and thus can only be solved through learning.

We start with the setting of NeRF [6] (recapped in Section IV-D) and replace the MLP converting coordinates and viewing direction into color and density with QTT-NF. Like Plenoxels [34], we choose a voxel grid resolution of $256^3$ and 28 channels to store 9 spherical harmonics (SH) coefficients per RGB channel and one voxel density value.

Next, we compare QTT-NF with a recent work, TensoRF [38], which likewise uses tensor decompositions, albeit triplanar ones. This work achieves remarkable reconstruction quality by using several state-of-the-art techniques on top of the proposed decomposition. We intentionally avoid such recipes that lead to better performance, voxel pruning, occupancy masks, extra losses (total variation), ray filtering, and progressive training schemes. One technique we borrow from that pool is the usage of a tiny MLP in place of SH for transforming voxel features into view-dependent color (NN), which retains the natural interpretability of the learned voxel grid features.

The standard testbed for neural rendering is the Blender dataset [6]. It consists of 8 synthetic 3D scenes, each with a hundred posed images of resolution $800 \times 800$. The scenes vary by complexity and include water and glossy surfaces, thus making it a challenging benchmark.

---

2Distributed under different Creative Commons licenses. Per scene: “chair”, “ficus”, “hotdog”, “materials”, “mic”: CC-0; “drums”: CC-BY; “lego”: CC-BY-NC; “ship”: CC-BY-SA.
We train neural rendering experiments for 80K steps with an LR schedule similar to the one from Section V-A, but with base LR $3\cdot3$, $R_{\text{max}} = 256$, 4096 rays per batch, and 512 uniform samples per ray. The code of all experiments is implemented in PyTorch [10]. The observed variance of image quality metrics over five runs is not large: \(\text{std}(\text{PSNR}) \sim 4\cdot e^{-2}\), \(\text{std}(\text{SSIM}) \sim 4\cdot e^{-4}\), \(\text{std}(\text{LPIPS}) \sim 5\cdot e^{-4}\). Results from Table II attest that QTT-NF is capable of reaching performance competitive with the prior art.

The ablation study of TT-rank, number of training steps, and SH basis in Fig. 7 shows that the results can be further improved, as most hyperparameters are not saturated.

The reconstruction quality of QTT-NF depends primarily on the TT-rank and scene complexity. This is contrary to TensoRF [38], which employs triplanar decompositions (CP, VM), and thus inevitably introduces a preference for axis-aligned content.

To demonstrate that, we introduce rotation of the scene around the Z-axis into the ground-truth poses of two scenes: “Lego” and “Hotdog”. The former scene contains many axis-aligned primitives, which favorably utilize triplanar decomposition rank. We ensured that no rotation angle resulted in content out of voxel grid bounds. As can be seen from Fig. 8 (top), rotations of the same axis-aligned scene by 15, 30, and 45 degrees result in significant performance drops with triplanar decomposition and all ranks, whereas the performance of QTT-NF drops more gracefully. An ordinary scene from Fig. 8 (bottom) demonstrates the vanishing of this effect due to the absence of axis alignment in any of the rotations.

The average performance of QTT-NF (with 2.16M parameters) is slightly higher than TensoRF (with 3.17M parameters). This attests to better parameter-efficiency of QTT-NF for scene representation.

C. Effect of Reduced Parameterization

We compare our reduced (v3) parameterization with the full one (v2) in the default QTT-NF setting with spherical harmonics and report results in Table III (top). As can be seen, the average PSNR does not change much depending on parameterization. This is contrary to the tensor denoising case described in Section V-A, which we can attribute to a different (longer) training regime.

![Fig. 8. Sensitivity of QTT-NF and TensoRF [38] to context-to-axes alignment. Scenes: Top – “Lego”, bottom – “Hotdog”. TensoRF exhibits a performance drop with all ranks when rotating axis-aligned scenes. Pattern: Dot – 25%, dash – 50%, dashdot – 75%, solid – 100% of the rank. Higher is better.](image-url)
TABLE III

| Metric     | Method (flavor)                  | Chair | Drums | Ficus | Hotdog | Lego | Materials | Mic | Ship | Avg. |
|------------|----------------------------------|-------|-------|-------|--------|------|-----------|-----|------|------|
| PSNR ↑     | QTT-NF (v2)                      | 31.88 | 24.95 | 30.72 | 35.30  | 32.17| 28.23     | 31.58| 27.84| 30.33|
|            | QTT-NF (v3)                      | 32.09 | 24.96 | 30.89 | 35.49  | 32.48| 28.22     | 31.50| 27.55| 30.40|
|            | Uncompressed                     | 32.54 | 25.10 | 31.30 | 33.53  | 31.91| 27.63     | 32.38| 22.76| 29.64|
|            | QTT-NF (init unc.)               | 32.26 | 25.09 | 30.93 | 35.53  | 32.76| 28.58     | 32.18| 27.93| 30.66|
| SSIM ↑     | QTT-NF (v2)                      | 0.953 | 0.918 | 0.967 | 0.970  | 0.956| 0.929     | 0.971| 0.843| 0.938|
|            | QTT-NF (v3)                      | 0.955 | 0.918 | 0.967 | 0.971  | 0.957| 0.929     | 0.971| 0.840| 0.939|
|            | Uncompressed                     | 0.967 | 0.925 | 0.970 | 0.960  | 0.959| 0.926     | 0.979| 0.785| 0.845|
|            | QTT-NF (init unc.)               | 0.958 | 0.921 | 0.968 | 0.972  | 0.961| 0.934     | 0.976| 0.845| 0.942|
| LPIPS ↓    | QTT-NF (v2)                      | 0.045 | 0.098 | 0.040 | 0.063  | 0.067| 0.093     | 0.041| 0.209| 0.085|
|            | QTT-NF (v3)                      | 0.062 | 0.095 | 0.039 | 0.060  | 0.066| 0.093     | 0.039| 0.203| 0.082|
|            | Uncompressed                     | 0.042 | 0.077 | 0.038 | 0.071  | 0.047| 0.088     | 0.025| 0.223| 0.076|
|            | QTT-NF (init unc.)               | 0.059 | 0.093 | 0.037 | 0.058  | 0.058| 0.087     | 0.031| 0.202| 0.078|

Initialization from uncompressed volume improves performance.

D. Initialization of QTT-NF Using TT-SVD

This set of experiments demonstrates the positive effect of QTT-NF initialization from an uncompressed voxel grid. We start by training such uncompressed representations with the same grid configuration (256 × 256 × 28 parameters). The training protocol is the same as for QTT-NF training, except for the usage of larger yet different learning rates for the parameters of spherical harmonics (1e-3) and density (1e-1). The results can be seen in Table III (Uncompressed); many scenes receive lower scores than training compressed representations from random initialization due to the lack of L1 or TV regularization. Further, we apply dimensions factorization of the uncompressed voxel grid to match the decomposition scheme from Fig. 3 and perform TT-SVD with rounding to the TT-rank matching our QTT-NF. The resulting decomposition is loaded into QTT-NF instead of random initialization per (2), and the same training protocol (only with a lower learning rate of 1e-3) is executed. The results of such fine-tuning are presented in Table III (QTT-NF init unc.). Evidently, TT-SVD provides a good initialization for QTT-NF, best in L2 distance between the approximation and the uncompressed volume, yet suboptimal in terms of the downstream task performance. This suboptimality is corrected through fine-tuning, and the resulting performance exceeds that of training from random initialization.

VI. DISCUSSION

1) Supported parallelism: All discussed sampling schemes support data parallelism: TT-NF parameters can be replicated across multiple computing devices, and a batch of samples is split among them. This is contrary to tensor contraction, where the main constraint is to be able to fit the entire uncompressed tensor into memory. On the other hand, optimized contraction schemes [55], [56] followed by subsampling are expected to outperform all sampling schemes starting with some sufficiently large batch size.

2) Low-level optimizations: The proposed sampling schemes (Algorithm 1 and 3) are designed to perform well at training time in all off-the-shelf deep learning frameworks with automatic differentiation support while keeping memory pressure low. Permutations enable the use of the standard Linear layer, which is one of the first functions undergoing heavy optimization on any new-generation computational device, along with memory operations such as permutations themselves, BLAS [57], convolutions [58], normalization layers [59], and activation functions [60]. It is possible, however, to get rid of permutations through a custom implementation of the BIMVP function (Algorithm 2), thus obtaining 2 × reduction in space requirements. CUTLASS [61] provides promising building blocks for implementing both forward and backward passes. Deploying TT-NF sampling for inference in production and edge devices can be feasible, provided the presence of AI accelerators optimized for neural network inference. An overview of the current state of mobile computing is given in [62].

3) Limitations: In short training protocols such as experiments with tensor denoising (Section V-A), v3 sampling consistently does not reach the performance of v2, despite effectively learning the same representation. We hypothesize that this effect may have an explanation in the deep linear networks literature. This effect is not pronounced in the NeRF setting with its longer training protocol, as seen in Table III.

VII. CONCLUSION

We presented TT-NF, a novel low-rank representation for neural fields that can be learned directly via backpropagation through samples and optimization. Our representation avoids instantiating the full uncompressed tensor but instead learns it in a compressed form by optimizing a non-convex objective defined by the target task. We applied TT-NF to a synthetic tensor denoising task, where we outperformed standard SVD-based approaches and the real-world novel view synthesis problem.
For the latter, we proposed QTT-NF, a 3D modification of TT-NF that handles hierarchical spaces, such as 3D voxel grids. Last but not least, we proposed efficient sampling algorithms for our neural fields and showed that these algorithms increase efficiency both w.r.t. speed and memory, making (Q)TT-NF friendly for applications such as rendering.

REFERENCES

[1] I. V. Oseledets, “Tensor-train decomposition,” SIAM J. Sci. Comput., vol. 33, no. 5, pp. 2295–2317, 2011.

[2] I. V. Oseledets and E. E. Tyrtyshnikov, “TT-cross approximation for multidimensional arrays,” Linear Algebra Appl., vol. 432, no. 1, pp. 70–88, 2010.

[3] Y. Zhou, A. R. Zhang, L. Zheng, and Y. Wang, “Optimal high-order tensor SVD via tensor-train orthogonal iteration,” IEEE Trans. Inf. Theory, vol. 68, no. 6, pp. 3991–4019, Jun. 2022.

[4] A. Novikov, A. Rodomanov, A. Oskin, and D. Vetrov, “Putting MRFs on a tensor train,” in Proc. 31st Int. Conf. Mach. Learn., PMLR, 2014, pp. 841–850.

[5] A. I. Boyko, M. Matrosov, I. Oseledets, D. Tsetserukou, and G. Ferrey, “TT-TSDF: Memory-efficient TSDF with low-rank tensor train decomposition,” in 2020 IEEE/RSJ Int. Conf. Intell. Robots Syst., 2020, pp. 10116–10121.

[6] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, “NeRF: Representing scenes as neural radiance fields for view synthesis,” in Proc. Eur. Conf. Comput. Vis., 2020, pp. 405–421.

[7] T. G. Kolda and B. W. Bader, “Tensor decompositions and applications,” SIAM Rev., vol. 51, no. 3, pp. 455–500, 2009.

[8] M. Usyutyakov, A. Makarova, R. Ballester-Ripoll, M. Rakhuba, A. Krause, and K. Schindler, “Cherry-picking gradients: Learning low-rank embeddings of visual data via differentiable cross-approximation,” in Proc. IEEE/CVF Int. Conf. Conf. Vis. Pattern Recognit., vol. 12, 2022, pp. 811–822.

[9] A. Paszke et al., “PyTorch: An imperative style, high-performance deep learning library,” in Proc. Nat. Conf. Mach. Learn., 2019, pp. 8024–8035.

[10] M. Iaderberg, A. Vedali, and A. Zisserman, “Speeding up convolutional neural networks with low rank expansions,” in Proc. Eur. Conf. Comput. Vis., 2022, pp. 333–350.

[11] I. V. Oseledets, “Tensor-train decomposition,” in Proc. Eur. Conf. Comput. Vis., 2020, pp. 405–421.

[12] J. Kossaifi, Y. Panagakis, A. Anandkumar, and M. Pantic, “Tensorly: Tensor learning in python,” J. Mach. Learn. Res., vol. 23, pp. 1–6, 2019.

[13] M. Usyutyakov, R. Ballester-Ripoll, and K. Schindler, “TnTensor: Tensor network learning with PyTorch,” J. Mach. Learn. Res., vol. 23, pp. 1–6, 2022.

[14] A. Novikov, P. Izmailov, V. Khurkulo, M. Figurnov, and I. Oseledets, “Tensor train decomposition on tensorflow (T3F),” in Proc. Eur. Conf. Comput. Vis., 2022, pp. 5855–5864.

[15] C. Sun, M. Sun, and H.-T. Chen, “Direct voxel grid optimization: Super-quadrics for real-time view surface reconstruction,” in Proc. Comput. Vis. - ECCV 2020, 2021, pp. 1207–1216.

[16] A. Yu, S. Fridovich-Keil, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa, “Plenoxels: Radiance fields without neural networks,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2022, pp. 5491–5500.

[17] T. Müller, A. Evans, C. Schied, and A. Keller, “Instant neural graphics primitives with a multiresolution hash encoding,” ACM Trans. Graph., vol. 41, no. 4, pp. 1–15, Jul. 2022.

[18] M. Jaderberg, A. Vedali, and A. Zisserman, “Speeding up convolutional neural networks with low rank expansions,” in Proc. Brit. Mach. Vis. Conf., 2014, [Online]. Available: https://bmva-archive.org.uk/bmvc/2014/papers/paper073/index.html

[19] V. Lebedev, Y. Ganin, M. Rakhuba, I. V. Oseledets, and V. S. Lempitsky, “Speeding up convolutional neural networks using fine-tuned CP-decomposition,” in Proc. Int. Conf. Learn. Representations, May 2015.

[20] A. Novikov, D. Podoprikhin, A. Oskin, and D. P. Vetrov, “Tensorizing neural networks,” in Proc. Annu. Conf. Neural Inf. Process. Syst., C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015, pp. 11406–11415.

[21] S. Fridovich-Keil, G. Meanti, F. R. Warburg, B. Recht, and A. Kanazawa, “Tensor field inference,” in Proc. Eur. Conf. Comput. Vis., 2022, pp. 5449–5459.

[22] A. Chen, Z. Xu, A. Geiger, J. Yu, and H. Su, “TensorFlow: Tensorial radiosity fields,” in Proc. Eur. Conf. Comput. Vis., 2022, pp. 33–350.

[23] A. Cao and J. Johnson, “HexPlane: A fast representation for dynamic scenes,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2023, pp. 130–141.

[24] S. Fridovich-Keil, G. Meanti, F. R. Warburg, B. Recht, and A. Kanazawa, “K-planes: Explicit radiance fields in space, time, and appearance,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2023, pp. 12479–12488.

[25] S. V. Dolgov, B. N. Khoromskij, I. V. Oseledets, and D. V. Savostyanov, “Computing extreme eigenvalues in higher dimensions using block tensor train format,” Comput. Phys. Commun., vol. 185, no. 4, pp. 1207–1216, 2014.

[26] V. I. Oseledets, “Approximation of matrices with logarithmic number of parameters,” Doklad. Math., vol. 80, no. 2, pp. 653–654, Oct. 2009.

[27] G. M. Morton, “Hierarchical organization of a file sequence,” Bell Syst. Tech. J., vol. 55, no. 5, pp. 420–436, May 1976.

[28] G. M. Morton, “Hierarchical organization of a file sequence,” Bell Syst. Tech. J., vol. 55, no. 5, pp. 420–436, May 1976.

[29] G. M. Morton, “Hierarchical organization of a file sequence,” Bell Syst. Tech. J., vol. 55, no. 5, pp. 420–436, May 1976.
Anton Obukhov received the Diploma in computational mathematics and cybernetics from Moscow State University, Moscow, Russia, in 2008, and the Ph.D. degree from Computer Vision Laboratory, ETH Zurich, Zurich, Switzerland, in 2022, supervised by Prof. Luc Van Gool and funded by the Toyota TRACE-Zurich Project. Between these degrees, he spent a decade working in the industry: he helped NVIDIA drive the adoption of NVIDIA CUDA technology in scientific computing and later helped Ubiquiti Networks build multiple video camera products with varying degree of artificial intelligence. He is currently a Postdoctoral Researcher with the Photogrammetry and Remote Sensing Lab, broadly interested in computer vision and machine learning research. In 2018, he joined the Computer Vision Laboratory, ETH Zurich.

Mikhail Usvyatsov received the bachelor’s degree in electrical engineering (with Hons.) from Ural Federal University, Yekaterinburg, Russia, the master’s degree (with Hons.) from the Moscow Institute of Physics and Technology, Dolgoprudny, Russia, and the Skolkovo Institute of Science and Technology, Moscow, and the Ph.D. degree from the Photogrammetry and Remote Sensing Lab, ETH Zurich, Switzerland, supervised by Prof. Konrad Schindler. He is currently a Software Engineer with Google working on efficient on-device computing. Before that, he spent one year in Innopolis, Kazan, where he moved from his hometown Ekaterinburg.

Konrad Schindler (Senior Member, IEEE) received the Diplomingenieur (M.Tech.) degree from the Vienna University of Technology, Vienna, Austria, in 1999, and the Ph.D. degree from the Graz University of Technology, Graz, Austria, in 2003. He was a Photogrammetric Engineer with the Private Industry and held researcher positions with the Graz University of Technology, Monash University, Melbourne, VIC, Australia, and ETH Zürich, Zürich, Switzerland. He was an Assistant Professor of image understanding with TU Darmstadt, Darmstadt, Germany, in 2009. Since 2010, he has been a tenured Professor of photogrammetry and remote sensing with ETH Zürich. His research interests include computer vision, photogrammetry, and remote sensing.

Luc Van Gool is currently a full professor of computer vision with ETH Zurich, Zurich, Switzerland, KU Leuven, Leuven, Belgium, and INSART, Sofia, Bulgaria. He leads research and teaches at all three institutions. He has authored more than 900 papers. His main interests include 3D reconstruction and modeling, object recognition, and autonomous driving. He has been the Program Committee Member of several major computer vision conferences (Program Chair ICCV’05, Beijing; the General Chair of ICCV’11, Barcelona; and ECCV’14, Zurich). He was the recipient of several best paper awards (David Marr Prize ’98, Best Paper CVPR’07). He was also the recipient of the Koenderink Award in 2016 and the “Distinguished Researcher” nomination by the IEEE Computer Society in 2017. In 2015, he received the five-yearly Excellence Prize from the Flemish Fund for Scientific Research. He was the holder of an ERC Advanced Grant (VarCity). He leads computer vision research for autonomous driving in the context of the Toyota TRACE labs at ETH and in Leuven and has an extensive collaboration with Huawei on the topic of image and video enhancement.