TT-TSDF: Memory-Efficient TSDF with Low-Rank Tensor Train Decomposition

Alexey I. Boyko[1], Mikhail P. Matrosov[2], Ivan V. Oseledets[1,3], Dzmitry Tsetserukou[2] and Gonzalo Ferrer[1]

***Abstract*— In this paper we apply the low-rank Tensor Train decomposition for compression and operations on 3D objects and scenes represented by volumetric distance functions. Our study shows that not only it allows for a very efficient compression of the high-resolution TSDF maps (up to three orders of magnitude of the original memory footprint at resolution of 512^3), but also allows to perform TSDF-Fusion directly in the low-rank form. This can potentially enable much more efficient 3D mapping on low-power mobile and consumer robot platforms.**

I. Introduction

Surface reconstruction from the depth sensor data is one of the key problems in robotics. There are several ways to represent an unstructured 3D map, including point clouds, voxel occupancy grids, triangular meshes, and octrees. Those representations propose different trade-offs in memory efficiency, scene precision, and available algorithms that can operate on them.

The majority of depth fusion algorithms rely on intermediate volumetric implicit scene representations such as Signed Distance Functions (sometimes called Signed Distance Fields), or their truncated (TSDF) versions as in the original TSDF Fusion paper by B. Curless and M. Levoy [1]. SDFs and TSDFs are functions existing in 3D space, and, therefore, must be stored in an appropriate 3D basis. A common approach is to use 3D piecewise-constant (voxel) basis. Voxelized SDFs and TSDFs are non-parametric volumetric representations of a 3D mesh with a high amount of redundancy, hence tensor decompositions might be a suitable tool for compressing it.

In this paper, we use Tensor Train decomposition (TT) [2]. It allows us to conduct many non-trivial operations on the volumetric map by manipulating directly with compressed representation, and it is even possible to build complex pipelines, such as KinectFusion [3].

Potentially it allows using commodity CPU-only embedded computers for large TSDF scene fusion in robotic 3D vision applications. A scene stored in TT-TSDF has 100-2000 times better memory efficiency for utilizing the compressed TSDF representation while having only 2-10 times computational overhead. In addition to that, GPU-based algorithms can also benefit from TT compression.

[1] The authors are with Skolkovo Institute of Science and Technology (Skoltech), CDISE department.

[2] Skoltech, Space Center department.

[3] Institute for Numerical Mathematics of Russian Academy of Sciences.
{alexey.boyko, mikhail.matrosov} @skolkovotech.ru
{i.oseledets, d.tsetserukou, g.ferrer} @skoltech.ru

This work was partially supported by the Ministry of Education and Science of the Russian Federation under Grant 14.756.31.0001.

Fig. 1. Visualisation of 512^3 TSDFs. Non-compressed TSDF and various degrees of TT-compression

Contributions of this paper are the following:

1) We applied TT decomposition on TSDFs of synthetic models. It has shown spectacular performance in the lossy compression, allowing up to $5 \cdot 10^{-4}$ of the original memory footprint for high-resolution volumetric data, while still providing very close reconstruction in terms of both metrics (IoU, Hausdorf or Chamfer) and visual appearance.
2) We have shown that it is possible to run TSDF-Fusion directly on the compressed form of the global volumetric map.

II. Related Work

Surface reconstruction from depth data has been an active research topic for the last decade. Previous works include the original TSDF Fusion [1], KinectFusion [3] (introduced real-time GPU processing), Kintinuous [4] (added iterative re-meshing), ElasticFusion [5] (optimized loop closures), BundleFusion [6] (sophisticated hierarchical algorithm that uses both SIFT descriptors and dense data processing).

All of these methods rely on voxelized representation

of 3D data, that has cubic requirements in both memory storage and complexity of operations. Those complications are widely recognized, and lead to a variety of algorithms trying to enhance data storage efficiency, such as Distance Field Compression [7] (lossy wavelet transform, up to 15% of original memory, or lossless transform, 25% percent), OctreeFusion [8] (Octree structure, $10-20\%$ of the original memory), Voxel Hashing [9] (hash tables instead of full arrays), and Directional SDF [10] (added direction information to the TSDF). Overall, the order of magnitude for compression in common learning-less methods is approximately 5-10 times, which roughly corresponds to the sparsity of TSDFs.

Another large branch of studies related to finding a low-parametric representation of 3D objects relies on neural networks to learn a mapping from 3D voxelized data to a smaller latent space, in the deterministic or probabilistic fashion [11], [12], [13]. Such approaches are good when a low-dimensional latent space is learned for a particular machine learning problem, such as classification or completion, but in general, they both require intensive computing capabilities and cannot handle arbitrary 3D scenes.

There is also a branch of algorithms for compressing mesh data, such as Google Draco [14], based on the Edgebreaker algorithm [15]. These algorithms are very effective (10^{-1} to 10^{-2}) for lossless compression of static meshed data. However, meshes are not widely used for 3D perception and 3D geometric learning problems, those for the most part rely on implicit volumetric data.

III. Tensor Train decomposition

A. Introduction

Low-rank tensor decomposition can be seen as a linear-algebraic way for lossy compression of a multi-dimensional array (tensor). There are different types of low-rank decompositions [16], the most stable of which are Tucker and Tensor Train [2]. Here we will use the second one.

Let us consider a 2D case first. Assume we have a function on a square grid $N \times N$ stored in a 2D array (a matrix A). We would like to find a matrix $\tilde{A}$ with deficient rank $R < N$ such that it has the smallest error in terms of Frobenius norm:

$$\epsilon = ||A - \tilde{A}||_F = \sqrt{\sum_{i,j}(a_{ij} - \tilde{a}_{ij})^2}. \tag{1}$$

According to the Eckart–Young–Mirsky theorem, the matrix $\tilde{A}$ is given by a truncated SVD decomposition of matrix A: $A = USV^* \implies \tilde{A} = U\tilde{S}V^*$. In addition, the square of the Frobenius norm error of this approximation is exactly equal to sum of squares of neglected singular values: $\epsilon = \sum_{i=R+1}^{N} \sigma_i^2$. Frobenius norm of error in matrices ϵ is (up a constant) equal to RMSE loss between the original function and its approximant on the grid. Therefore with the SVD-truncated approximation, we found the best (in terms of RMSE) low-rank discrete functional approximation with a controllable error. A similar logic can be applied for arbitrary d-dimensional arrays, and this is the core idea behind Tensor Train decomposition.

B. Definition and complexities

Let us consider a d-dimensional array $F(i_1, \dots, i_d)$ with sizes of modes $N_1, \dots, N_d$. The following representation of this array is called a Tensor Train:

$$\begin{aligned} F(i_1, i_2, \dots, i_d) = \sum_{m_1, m_2, \dots, m_{d-1}}^{r_1, r_2, \dots, r_{d-1}} G^{(1)}(i_1, m_1) \cdot \\ \cdot G^{(2)}(m_1, i_2, m_2) \cdot \dots \cdot G^{(d)}(m_{d-1}, i_d), \end{aligned} \tag{2}$$

where arrays $G^{(k)}$ are called TT-cores, and numbers r_k are called TT-ranks.

The standard way of obtaining cores $G^{(k)}$ from a given array is to use the TT-SVD algorithm [2]. Its key idea is sequential reshaping of the multidimensional array into rectangular matrices, with sequential factorizing of dimensions using truncated SVD.

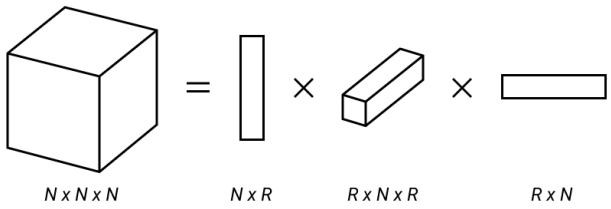


Fig. 2. TT-decomposition of a tensor of size $N \times N \times N$ with rank R.

Along with the reduction in memory, Tensor Train representation allows performing mathematical operations such as $+, -, *$, Hadamard, Kronecker and dot products, summation over any index, directly in the low-rank form. Additionally one can perform error-controllable compression and re-compression using TT-SVD [2], application of linear operators with low-rank TT-matrices and application of nonlinear operators to compressed data without full decompression using TT-CROSS or DMRG/ALS/AMEn algorithms [17], while still allowing to perform deep learning-style backpropagation. All this allows embedding compressed tensors in a variety of algorithms from scientific computing and data science. Also, it has excellent code availability (`TT-Toolbox`, `ttpy`, `tntorch`, `t3f`, `tensorly` libraries).

In the case of 3D data the total memory required by TT is $N_1 r_1 + r_1 N_2 r_2 + N_3 r_2 = O(NR^2)$ instead of $O(N^3)$. Therefore results are the best for high grid resolutions as compression ratio is $O(\frac{R^2}{N^2})$. Here $R = \max(r_i)$ and $N = \max(N_i)$.

The complexity of compressing a d-dimensional array into a rank-R TT-approximation using TT-SVD is $O(N^d R^2)$. The complexity of adding two TTs with TT-ranks R and re-compressing the sum back into a rank-R tensor is $O(dNR^3)$.

IV. Compression of SDF and TSDF

In this paragraph, we analyze how low-rank tensor compression affects Signed Distance Field and its truncated version. Given that both are widely used in the 3D community it is natural to compare their behaviour under compression.

Let us consider a closed 3D model Ω with surface $\partial\Omega$. For any 3D point $p \in \mathbb{R}^3$ Signed Distance Function (Field) is defined as:

$$SDF(p) = \begin{cases} dist(p, \partial\Omega), & \text{if outside of object } \Omega \\ -dist(p, \partial\Omega), & \text{otherwise.} \end{cases} \tag{3}$$

Truncated SDF is defined as:

$$TSDF(p) = \begin{cases} \mu, & \text{if } \mu \leq SDF(p) \\ SDF(p), & \text{if } -\mu < SDF(p) < \mu \\ -\mu, & \text{if } SDF(p) \leq -\mu. \end{cases} \tag{4}$$

It is well known in the tensor community that SVD-based approximations struggle to find good approximations when it comes to discontinuous or irregular data. However, the main observation of this paper is that although low-rank TSDF representation is erroneous (in terms of RMSE), the zero-surface of the function is still of a sufficient quality to produce good reconstructions of the mesh. On the contrary, the low-rank SDF representation, although a priori looks highly redundant and hence appropriate for TT compression, yields poor results.

We illustrate this effect on Figure 3. We take a watertight model of a plane and compute both true SDF (second subfigure) and TSDF (fourth subfigure). Middle crossections are shown. While both of those functions work well to represent the surface of the model as zero isosurface, it is not the case for their compressed forms. With TT-compression with maximum ranks $R = 7$ SDF is unable to provide a connected model, while TSDF still visually corresponds to the correct shape of the crossection of the plane.

This is possibly because TT-SVD minimizes RMSE for the whole volume thus wasting singular vectors on storing values in voxels far from the zero-surface - the information that is not useful for reconstructing 3D mesh from its voxelized representation.

V. TSDF Fusion and its compressed form

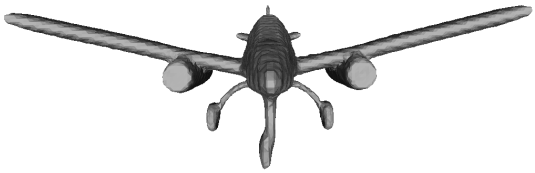
TSDF Fusion is a family of online algorithms [1], [3], [4], [5], [6] to build a global volumetric map of a 3D scene using a series of depth maps.

As the proof of concept we propose a modified KinectFusion algorithm, where mixing of separate TSDF volumetric frames into the global volumetric map is done fully in the compressed format.

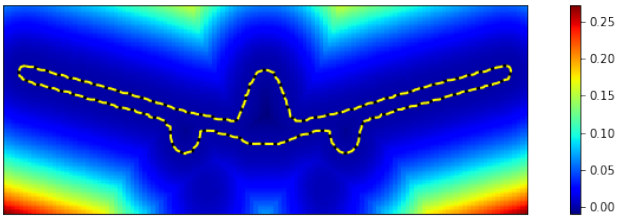
1) Obtain and preprocess a depth image,
2) estimate camera pose,
3) construct a TSDF frame of the current depth image (Alg. 1),
4) compress the TSDF frame using TT-SVD,
5) fuse TT-TSDF frame into the compressed global volumetric map (5).

Technically it is possible to construct the current TSDF frame from a depth image directly in the TT format using an algorithm from the TT-CROSS [17] family. However both the theory and our experiments with Amen-CROSS have shown that the resulting TT frame is of full rank ($R = N$), requiring even more memory than an uncompressed tensor, while being sufficiently slower to obtain ($O(dNR^3) = O(N^4)$ per frame instead of $O(N^3)$). And then we would still need to run TT-SVD to re-compress it into low rank. Therefore it is hard to avoid dealing with one full TSDF frame every iteration.

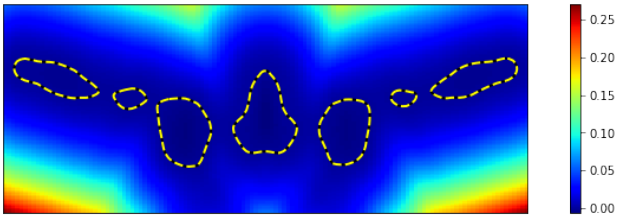
Camera pose estimation is out of the scope of this paper, but it can be done by a variety of methods, using matching depth point clouds [3] or sparse descriptors [6]. For the proof of concept we considered a camera pose as given for every frame.



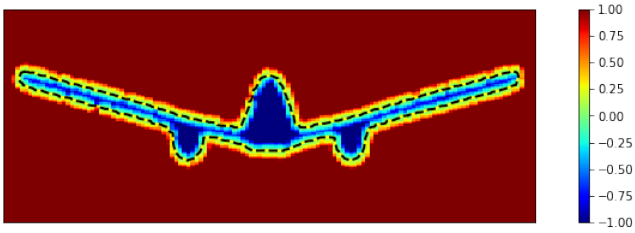
ModelNet `plane-0629` (watertight version)



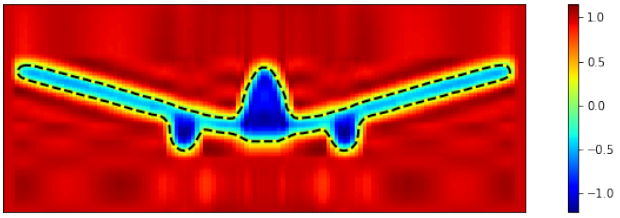
Crossection of the SDF (uncompressed)



Crossection of the TT-SDF ($R = 7$). Original surface is far from being well reconstructed after compression.



Crossection of the TSDF (uncompressed)



Crossection of the TT-TSDF ($R = 7$). Original surface is well reconstructed.

Fig. 3. The model and middle crossections of its SDF and TSDF. Uncompressed forms and $R = 7$ TT-forms are shown. Dashed line corresponds to the surface of the model (zero isolevel)

Depth images are preprocessed with an inpainting algorithm by A.Telea [18]. This is done to avoid propagation of sharp discontinuities and aliasing in the data that is to be put into TT-compression.

Given a preprocessed depth image D with size of $W \times H$ and a camera pose matrix T, we construct the corresponding TSDF of the depth frame according to the standard algorithm (see Alg. 1). $\lfloor \dots \rceil$ denotes rounding to the closest integer. If `pix_x` or `pix_y` are beyond the depth image then we put distance $d = 0$. This part of the algorithm requires $O(N^3)$ operations per depth image.

Data: depth image D, camera pose matrix T
Result: $TSDF_{frame}$
for *all voxel centers* $\tilde{p}$ *in volume* V **do**
  $(x, y, z) := T^{-1}\tilde{p}$,
  `pix_x, pix_y` := $\lfloor x/z \rceil, \lfloor y/z \rceil$,
  $d := D($`pix_x, pix_y`$)$,
  TSDF($\tilde{p}$) := truncate$(d - z, \pm\mu)$
end

Algorithm 1: Depth image to volumetric TSDF frame

The fusion is done in the TSDF-Fusion / KinectFusion fashion:

$$\begin{aligned} F_k(p) &= F_{k-1}(p) + W_{frame} TSDF_{frame}, \\ W_k(p) &= W_{k-1}(p) + W_{frame}, \end{aligned} \tag{5}$$

where W_{frame} is the binary view frustum mask of the current frame.

One minor difference with the original algorithm is that after obtaining a non-compressed 3D TSDF for a piece of the 3D scene generated from the current depth image we then compress the full update term $W_{frame} TSDF_{frame}$ by applying truncated TT-SVD with a fixed maximum rank R. It requires $O(N^3 R^2)$ operations, which is the most computationally expensive part. Then we further perform fusion to the compressed global map $F_k(p)$. The second minor difference is that instead of using running average over frames we separately accumulate the enumerator $F_k(p)$ and denominator $W_k(p)$. The final TSDF is then computed as $F_k(p)/W_k(p)$ and the mesh of the scene is restored using the Marching Cubes algorithm [19].

VI. Evaluation

A. Synthetic models

In this section we will test the impact of our Tensor Train compression on TSDFs. To analyze the behavior of the meshes we will test on Stanford Dragon, Armadillo and selected models from ModelNet40 [20] and ShapeNet [21]. Those are popular synthetic datasets for 3D objects. Many of the models have fine details and complex non-convex geometries, so it is informative for testing lossy compression algorithms.

There are some fundamental limitations that are not related to our compression. The first one is that we use TSDF-Fusion/KinectFusion pipeline and we can only see the external shell of each model. And statistical nature of the KinectFusion better detects features that can be observed from multiple viewpoints. The second one is that a minimum size of fine details is limited by the voxel size. The third one is that we recover mesh from TSDF by the Marching Cubes, and it adds angular artifacts to the surface. The original KinectFusion paper [3] as well as DeepSDF paper [13] render the surface of the recovered model by directly ray-tracing through TSDF, nevertheless, for our purposes it is not necessary.

For gathering synthetic depth data from the meshed models we use an approach similar to the one used in [13]. We normalize a model isotropically such that it fits in a unit sphere with 5% margin. Then we create 100 uniformly distributed camera observation points, render virtual depth maps and perform standard KinectFusion on the pairs (depth, camera pose) data with resolution 512^3 voxels. Meshes are recovered using the Marching Cubes algorithm [19]. This approach produces clean watertight meshes with no interior structure, even from relatively dirty non-manifold meshes.

For quantitative evaluation, we will be using Intersection over Union metric (Hausdorff (Table II) and Chamfer distance (Fig. 5) metric on the recovered watertight mesh.

B. Intersection over Union (IoU) metric

For two closed 3D objects (Ω and Θ) IoU is defined as:

$$IoU(\Omega, \Theta) = \frac{|\Omega \cap \Theta|}{|\Omega \cup \Theta|}.$$

IoU is a volume-based metric, and it is naturally computed on the binary masks of bodies where $TSDF < 0$. IoU is the most straightforward metric to compute, especially considering that mesh-related metrics require cleaning mesh models by running high-resolution TSDF-Fusion beforehand.

C. Hausdorff distance

For a pair of surfaces $\partial\Omega, \partial\Theta$ Hausdorff distance is:

$$d_H(\partial\Omega, \partial\Theta) = \inf_{\epsilon > 0} \left[\partial\Omega \subseteq \partial\Theta_\epsilon \quad \text{and} \quad \partial\Theta \subseteq \partial\Omega_\epsilon \right].$$

In our ablation study we report relative (dimensionless) Hausdorff distance, which is d_H normalized by the length of the diagonal of the bounding box of the mesh.

D. Chamfer distance

Symmetric Chamfer distance is defined as:

$$d_{CD}(\partial\Omega, \partial\Theta) = \sum_{x \in \partial\Omega} \min_{y \in \partial\Theta} ||x - y||_2^2 + \sum_{y \in \partial\Theta} \min_{x \in \partial\Omega} ||x - y||_2^2.$$

In our test in Fig. 5 we measure symmetric Chamfer distance on 30000 points. On the plot we report $d_{CD}/30000$.

Note that we measure Hausdorff and Chamfer distances between non-compressed TSDF-recovered mesh and every of TT-compressed TSDF-recovered meshes. This is to negate an uncertainty related to non-manifoldness and possibly existing interiors of 3D meshes from evaluated datasets.

Visual results on the quality of shapes recovered from compressed TSDFs are shown in Figures 4 and 5. As seen from both pictures and metrics, small or thin details start

deteriorating first as the rank decreases. The best results are obtained for the chair model (simple shape, many smooth surfaces), and the worst are for the Stanford Dragon and Stanford Armadillo (complex shapes, a lot of thin details).

TABLE I

IOU METRIC (LARGER IS BETTER)

Max-Rank	40	30	20	10
Car	.9820	.9720	.9580	.8803
Plane	.9799	.9730	.9608	.9131
Dragon	.9831	.9749	.9543	.8814
Bench	.9758	.9706	.9565	.8890
Chair	.9828	.9800	.9701	.9097

TABLE II

RELATIVE HAUSDORFF DISTANCE $\times 10^3$ (SMALLER IS BETTER)

Max-Rank	40	30	20	10
Car	$.28 \pm .41$	$.38 \pm .41$	$.57 \pm .79$	1.41 ± 1.95
Plane	$.19 \pm .29$	$.26 \pm .41$	$.39 \pm .61$	$.95 \pm 1.35$
Dragon	$.42 \pm .56$	$.70 \pm 1.17$	1.34 ± 2.42	4.4 ± 7.3
Bench	$.14 \pm .19$	$.18 \pm .26$	$.30 \pm .43$	$.82 \pm 1.30$
Chair	$.12 \pm .16$	$.14 \pm .20$	$.24 \pm .33$	$.79 \pm 1.13$

E. Depth dataset

In this section we will test the fusion algorithm discussed in the section V.

As shown on our tests on synthetic models (see Fig. 4), maximum rank $R = 40$ is sufficient to preserve fine details such as car wheel rims or dragon's horns. Our KinectFusion implementation maintains a fixed maximum TT-rank $R = 40$ for the global TSDF map. Only a single uncompressed volumetric TSDF frame needs to be stored in the memory. It then can be compressed and fused with the global map directly in a compressed form. The experiments were conducted on the global volumetric grid of resolution $324 \times 209 \times 231$.

We use a dataset from Zeng [22] that contains RGBD frames (we use depth channel only), intrinsic calibration matrix as well as extrinsic transformation matrices for every frame. Evaluating camera positions is out of the scope of this work, and we assume extrinsic camera matrices are already known. For the scene from the dataset, metrics are shown in Table III. Reconstructed scenes are shown in Fig. 6. As seen from the pictures and the Hausdorff distance, TT is an effective method for compressing TSDF.

VII. CONCLUSIONS

We have proposed a new algorithm for handling TSDFs and performing fusion of a global TSDF map in a compressed form using low-rank Tensor Train decomposition. It is learning-less, mathematically simple and relies only on a single hyperparameter - desired truncation rank.

We have shown that since Tensor Train allows for algebraic computations in a compressed format, it may be directly embedded in a KinectFusion pipeline, such that the global map is always maintained in the compressed form with a fixed low rank. We have shown that it also works for sophisticated synthetic models with fine details. For the resolution of $N = 512$ it performs very efficiently, allowing for storing high-resolution 3D TSDFs requiring only for 0.64% of the memory for an almost visually indistinguishable quality compression ($R = 40$), and up to 0.046% for the compression that still preserves main geometric features of the model ($R = 10$).

The future directions of this work may be to use differentiability property of the Tensor Train representation to apply it as a learning-less implicit TSDF shape embedding layer for 3D deep learning tasks, such as shape completion.

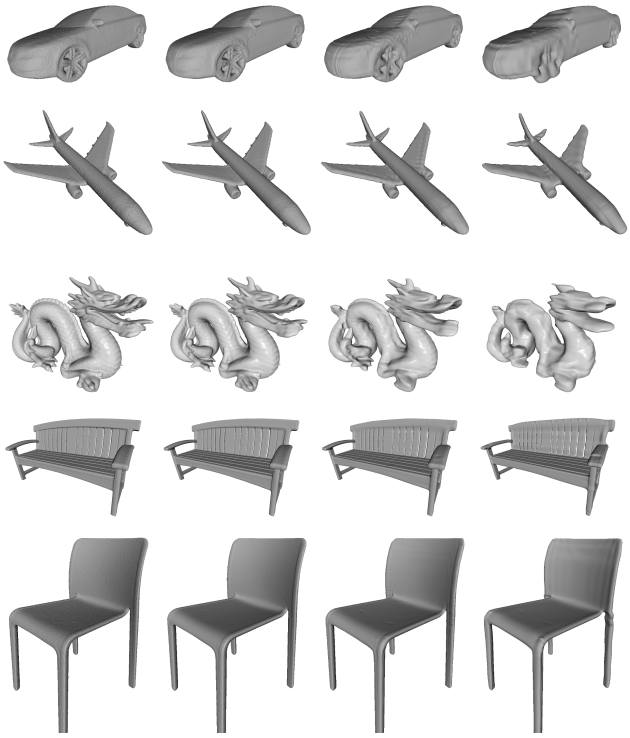


Fig. 4. *From left to right*: Uncompressed TSDF, Low-rank TT-TSDFs with maximum tensor ranks $R = 40$ (0.64% memory), $R = 20$ (0.17% memory), $R = 10$ (0.046% memory)

TABLE III

TENSORIZED TSDF FUSION METRICS

	KinectFusion [3], CPU	TT-TSDF Fusion (ours), CPU
Hausdorff distance, relative	0	$2.3 \pm 0.5 \times 10^{-2}$
Memory for scene	62.6 MB 100%	1.43 MB 2.3%
Integration time per frame	42 ms	1734 ms
Total time per frame	1.26 s	2.94 s

REFERENCES

[1] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," *SIGGRAPH*, pp. 303–312, 1996.

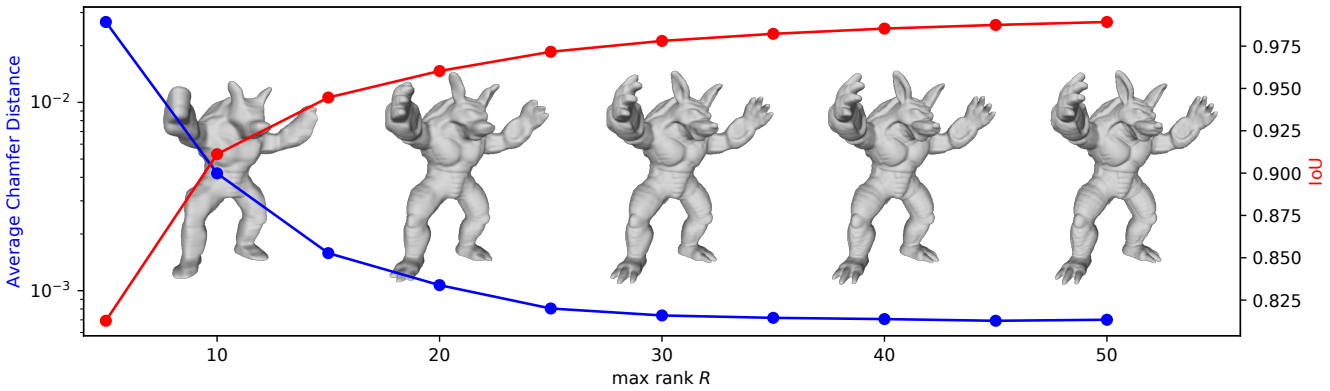
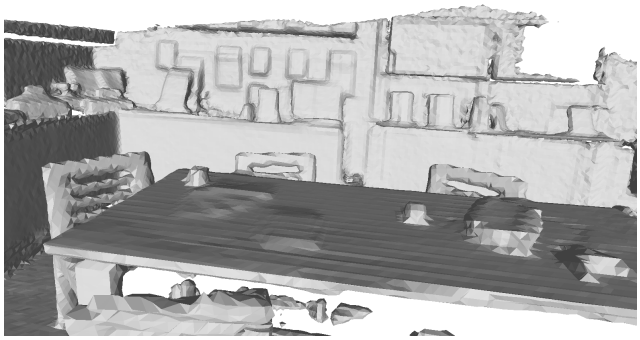
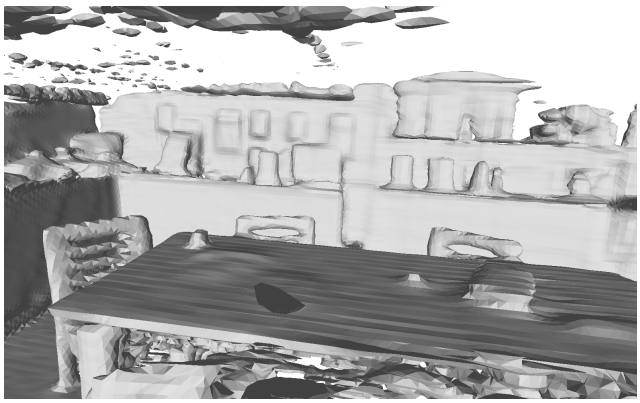


Fig. 5. Metrics and corresponding model for 512^3 TSDF of Stanford Armadillo depending on R.



Standard KinectFusion



TSDF-Fusion in TT-compressed form with maximum tensor rank $R = 40$

Fig. 6. Comparison of the full and the compressed fusion

[2] I. V. Oseledets, "Tensor-Train Decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.

[3] R. Newcombe, A. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "KinectFusion: Real-Time Dense Surface Mapping and Tracking," in *IEEE International Symposium on Mixed and Augmented Reality*, pp. 127–136, 2011.

[4] T. Whelan, M. Kaess, and M. Fallon, "Kintinuous: Spatially extended kinectfusion," *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, p. 7, 2012.

[5] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison, "ElasticFusion: Dense SLAM without a pose graph," *Robotics: Science and Systems*, vol. 11, 2015.

[6] A. Dai, M. Nieundefinedner, M. Zollhöfer, S. Izadi, and C. Theobalt, "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration," *ACM Trans. Graph.*, vol. 36, May 2017.

[7] M. W. Jones, "Distance field compression," *Journal of WSCG*, vol. 12, no. 2, pp. 199–204, 2004.

[8] M. Zeng, F. Zhao, J. Zheng, and X. Liu, "Octree-based fusion for realtime 3D reconstruction," *Graphical Models*, vol. 75, no. 3, pp. 126–136, 2013.

[9] M. Niesner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3D reconstruction at scale using voxel hashing," *ACM Transactions on Graphics*, vol. 32, no. 6, 2013.

[10] M. Splietker and S. Behnke, "Directional tsdf: Modeling surface orientation for coherent meshes," 11 2019.

[11] C. Hane, S. Tulsiani, and J. Malik, "Hierarchical surface prediction for 3D object reconstruction," in *3DV*, pp. 412–420, 2017.

[12] Z. Chen and H. Zhang, "Learning implicit fields for generative shape modeling," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2019-June, pp. 5932–5941, 2019.

[13] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation," in *CVPR*, 2019.

[14] "Draco: a library for compressing and decompressing 3d geometric meshes and point clouds." https://github.com/google/draco.

[15] J. Rossignac, "3d compression made simple: Edgebreaker with zipand-wrap on a corner-table," pp. 278 – 283, 06 2001.

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

[17] I. Oseledets and E. Tyrtyshnikov, "TT-cross approximation for multidimensional arrays," *Linear Algebra and Its Applications*, vol. 432, no. 1, pp. 70–88, 2010.

[18] A. Telea, "An image inpainting technique based on the fast marching method," *J. Graphics, GPU, Game Tools*, vol. 9, pp. 23–34, 2004.

[19] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *SIGGRAPH Comput. Graph.*, vol. 21, p. 163–169, Aug. 1987.

[20] Zhirong Wu, S. Song, A. Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1912–1920, June 2015.

[21] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "Shapenet: An information-rich 3d model repository," 2015. cite arxiv:1512.03012.

[22] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser, "3dmatch: Learning local geometric descriptors from rgb-d reconstructions," in *CVPR*, 2017.