Real-time Compression and Streaming of 4D Performances

DANHANG TANG, MINGSONG DOU, PETER LINCOLN, PHILIP DAVIDSON, KAIWEN GUO, JONATHAN TAYLOR, SEAN FANELLO, CEM KESKIN, ADARSH KOWDLE, SOFIEN BOUAZIZ, SHAHRAM IZADI, and ANDREA TAGLIASACCHI, Google Inc.

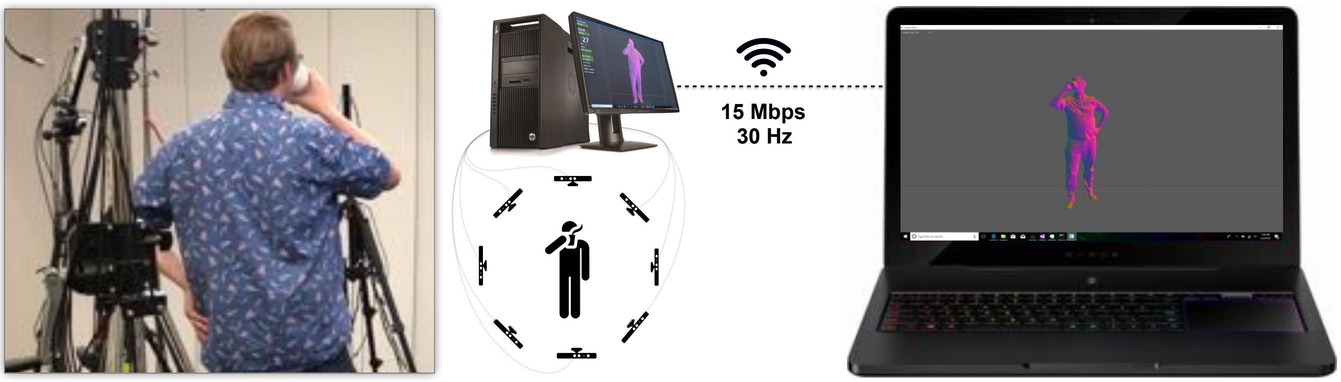

Fig. 1. We acquire a performance via a rig consisting of 8× RGBD cameras in a panoptic configuration. The 4D geometry is reconstructed via a state-of-the-art non-rigid fusion algorithm, and then compressed in real-time by our algorithm. The compressed data is streamed over a medium bandwidth connection (≈ 20Mbps), and decoded in real-time on consumer-level devices (Razer Blade Pro w/ GTX 1080m graphic card). Our system opens the door towards real-time telepresence for Virtual and Augmented Reality – see the accompanying video.

We introduce a realtime compression architecture for 4D performance capture that is two orders of magnitude faster than current state-of-the-art techniques, yet achieves comparable visual quality and bitrate. We note how much of the algorithmic complexity in traditional 4D compression arises from the necessity to encode geometry using an *explicit* model (i.e. a triangle mesh). In contrast, we propose an encoder that leverages an *implicit* representation (namely a Signed Distance Function) to represent the observed geometry, as well as its changes through time. We demonstrate how SDFs, when defined over a small local region (i.e. a block), admit a low-dimensional embedding due to the innate geometric redundancies in their representation. We then propose an optimization that takes a Truncated SDF (i.e. a TSDF), such as those found in most rigid/non-rigid reconstruction pipelines, and efficiently projects each TSDF block onto the SDF latent space. This results in a collection of low entropy tuples that can be effectively quantized and symbolically encoded. On the decoder side, to avoid the typical artifacts of block-based coding, we also propose a variational optimization that compensates for quantization residuals in order to penalize unsightly discontinuities in the decompressed signal. This optimization is expressed in the SDF latent embedding, and hence can also be performed efficiently. We demonstrate our compression/decompression architecture by realizing, to the best of our knowledge, the first system for streaming a real-time captured 4D performance on consumer-level networks.

Authors' address: Danhang Tang; Mingsong Dou; Peter Lincoln; Philip Davidson; Kaiwen Guo; Jonathan Taylor; Sean Fanello; Cem Keskin; Adarsh Kowdle; Sofien Bouaziz; Shahram Izadi; Andrea Tagliasacchi, Google Inc.

0730-0301/2018/11-ART256
https://doi.org/10.1145/3272127.3275096

CCS Concepts: • **Theory of computation → Data compression**; • **Computing methodologies → Computer vision**; *Volumetric models*;

Additional Key Words and Phrases: 4D compression, free viewpoint video.

ACM Reference Format:
Danhang Tang, Mingsong Dou, Peter Lincoln, Philip Davidson, Kaiwen Guo, Jonathan Taylor, Sean Fanello, Cem Keskin, Adarsh Kowdle, Sofien Bouaziz, Shahram Izadi, and Andrea Tagliasacchi. 2018. Real-time Compression and Streaming of 4D Performances. *ACM Trans. Graph.* 37, 6, Article 256 (November 2018), 11 pages. https://doi.org/10.1145/3272127.3275096

1 INTRODUCTION

Recent commercialization of AR/VR headsets, such as the Oculus Rift and Microsoft Hololens, has enabled the consumption of 4D data from unconstrained viewpoints (i.e. free-viewpoint video). In turn, this has created a demand for algorithms to capture [Collet et al. 2015], store [Prada et al. 2017], and transfer [Orts-Escolano et al. 2016] 4D data for user consumption. These datasets are acquired in capture setups comprised of hundreds of traditional color cameras, resulting in computationally intensive processing to generate models capable of free viewpoint rendering. While pre-processing can take place offline, the resulting "temporal sequence of colored meshes" is far from compressed, with bitrates reaching ≈ 100Mbps. In comparison, commercial solutions for internet video streaming, such as Netflix, typically cap the maximum bandwidth at a modest 16Mbps. Hence, state-of-the-art algorithms focus on *offline* compression, but with computational costs of 30s/frame on a cluster comprised of ≈ 70 high-end workstations [Collet et al. 2015], these solutions will be a struggle to employ in scenarios beyond a professional capture studio. Further, recent advances in VR/AR telepresence not only require the transfer of 4D data over *limited bandwidth* connections to a client with *limited compute*, but also require this to happen in a *streaming* fashion (i.e. real-time compression).

In direct contrast to Collet et al. [2015], we approach these challenges by revising the 4D pipeline end-to-end. First, instead of requiring $\approx$ 100 RGB and IR cameras, we only employ a dozen RGB+D cameras, which are sufficient to provide a panoptic view of the performer. We then make the fundamental observation that much of the algorithmic/computational complexity of previous approaches revolves around the use of triangular meshes (i.e. an *explicit* representation), and their reliance on a UV parameterization for texture mapping. The state-of-the-art method by Prada et al. [2017] encodes differentials in topology (i.e. connectivity changes) and geometry (i.e. vertex displacements) with mesh operations. However, substantial efforts need to be invested in the maintenance of the UV atlas, so that video texture compression remains effective. In contrast, our geometry is represented in *implicit* form as a truncated signed distance function (TSDF) [Curless and Levoy 1996]. Our novel TSDF encoder achieves geometry compression rates that are lower than those in the state of the art (7 Mbps vs. 8 Mbps of [Prada et al. 2017]), at a framerate that is two orders of magnitude faster than current mesh-based methods (30 fps vs. 1 min/frame of [Prada et al. 2017]). This is made possible thanks to the implicit TSDF representation of the geometry, that can be efficiently encoded in a low dimensional manifold.

2 RELATED WORKS

We give an overview of the state-of-the-art on 3D/4D data compression, and highlight our contributions in contrast to these methods.

Mesh-based compression. There is a significant body of work on compression of triangular meshes; see the surveys by Peng et al. [2005] and by Maglo et al. [2015]. Lossy compression of a static mesh using a Fourier transform and manifold harmonics is possible [Karni and Gotsman 2000; Lévy and Zhang 2010], but these methods hardly scale to real-time applications due to the computational complexity of spectral decomposition of the Laplacian operator. Furthermore, not only does the connectivity need to be compressed and transferred to the client, but lossy quantization typically results in low-frequency aberrations to the signal [Sorkine et al. 2003]. Under the assumption of temporally coherent topology, these techniques can be generalized to compression of animations [Karni and Gotsman 2004]. However, quantization still results in low-frequency aberrations that introduce very significant visual artifacts such as *foot-skate* effects [Vasa and Skala 2011]. Overall, all these methods, including the [MPEG4/AFX 2008] standard, assume temporally consistent topology/connectivity, making them unsuitable for the scenarios we seek to address. Another class of geometric compressors, derived from the EdgeBreaker encoders [Rossignac 1999], lie at the foundation of the modern Google/Draco library [Galligan et al. 2018]. This method is compared with ours in Section 4.

Compression of volumetric sequences. The work by Collet et al. [2015] pioneered volumetric video compression, with the primary focus of compressing hundreds of RGB camera data in a single texture map. Given a keyframe, they first non-rigidly deform this template to align with a set of subsequent subframes using embedded deformations [Sumner et al. 2007]. Keyframe geometry is compressed via vertex position quantization, whereas connectivity information is delta-encoded. Differentials with respect to a linear motion predictor are then quantized and then compressed with minimal entropy. Optimal keyframes are selected via heuristics, and post motion-compensation residuals are implicitly *assumed* to be zero. As the texture atlas remains constant in subframes (the set of frames between adjacent keyframes), the texture image stream is temporally coherent, making it easy to compress by traditional video encoders. As reported in [Prada et al. 2017, Tab.2], this results in texture bitrates ranging from 1.8 to 6.5 Mbps, and geometry bitrates ranging from 6.6 to 21.7 Mbps. The work by Prada et al. [2017] is a direct extension of [Collet et al. 2015] that uses a differential encoder. However, these differentials are *explicit mesh editing* operations correcting the residuals that motion compensation failed to explain. This results in better texture compression, as it enables the atlases to remain temporally coherent across entire sequences (-13% texture bitrate), and better geometry bitrates, as only mesh differentials, rather than entire keyframes, need to be compressed (-30% geometry bitrate); see [Prada et al. 2017, Tab.2]. Note how neither [Collet et al. 2015] nor [Prada et al. 2017] is suitable for streaming compression, as they both have a runtime complexity that is in the order of 1 min/frame.

Volume compression. Implicit 3D representation of a volume, such as signed distance function (SDF), has also been explored for 3D data compression. The work in [Schelkens et al. 2003] is an en example based on 3D wavelets that form the foundation of the JPEG2000/jp3D standard. Dictionary learning and sparse coding have been used in the context of local surface patch encoding [Ruhnke et al. 2013], but the method is far from real-time. Similar to this work, Canelhas et al. [2017] chose to split the volume into blocks, and perform KLT compression to efficiently encode the data. In their method, discontinuities at the TSDF boundaries are dealt with by allowing blocks to overlap, resulting in very high bitrate. Moreover, the encoding of a single slice of $\times100$ blocks (10% of volume) of size 16^3 takes $\approx 45ms$, making the method unsuitable for real-time scenarios.

Compression as 3D shape approximation. Shape approximation can also be interpreted as a form of lossy compression. In this domain we find several variants, from traditional decimation [Cohen-Steiner et al. 2004; Garland and Heckbert 1997] to its level-of-detail variants [Hoppe 1996; Valette and Prost 2004]. Another form of compression replaces the original geometry through a set of approximating proxies, such as bounding cages [Calderon and Boubekeur 2017] and sphere-meshes [Thiery et al. 2013]. The latter are somewhat relevant due to their recent extension to 4D geometry [Thiery et al. 2016], but [MPEG4/AFX 2008], only support temporally coherent topology.

Contributions. In summary, our main contribution is a novel end-to-end architecture for real-time 4D data streaming on consumer-level hardware, which includes the following elements:

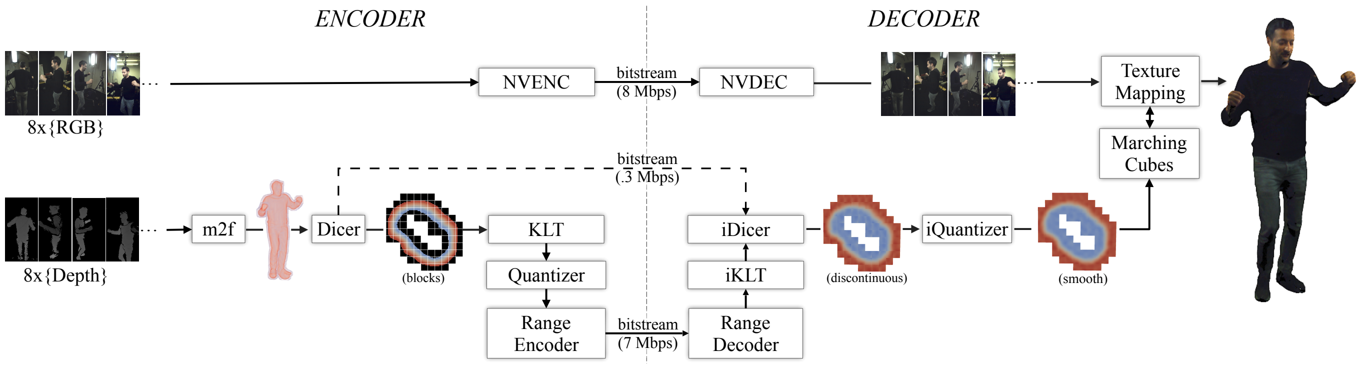

Fig. 2. The architecture of our real-time 4D compression/decompression pipeline; we visualize 2D blocks for illustrative purposes only. From the depth images, a TSDF volume is created by motion2fusion [Dou et al. 2017]; each of the $8 \times 8 \times 8$ blocks of the TSDF is then embedded in a compact latent space, quantized and then symbolically encoded. The decompressor inverts this process, where an optional inverse quantization (iQ) filter can be used to remove the discontinuities caused by the fact that the signal compresses adjacent blocks *independently*. A triangular mesh is then extracted from the decompressed TSDF via iso-surfacing. Given our limited number of RGB cameras, the compression of color information is performed per-channel and *without* the need of any UV mapping; given the camera calibrations, the color streams can then be mapped on the surface projectively within a programmable shader.

- A new efficient compression architecture leveraging implicit representations instead of triangular meshes.
- We show how we can reduce the number of coefficients needed to encode a volume by solving, at run-time, a collection of small least squares problems.
- We propose an optimization technique to enforce smoothness among neighboring blocks, removing visually unpleasant block-artifacts.
- These changes allow us to achieve state-of-the-art compression performance, which we evaluate in terms of rate vs distortion analysis.
- We propose a GPU compression architecture, that, to the best of our knowledge, is the first to achieve real-time compression performance.

3 TECHNICAL OUTLINE

We overview our compression architecture in Figure 2. Our pipeline assumes a stream of depth and color image sets $\{\mathcal{D}_t^n, \mathcal{C}_t^n\}$ from a multiview capture system that are used to generate a volumetric reconstruction (Section 3.1). The encoder/decoder has two streams of computation executed in parallel, one for geometry (Section 3.2), one for color (Section 3.5), which are transferred via a peer-to-peer network channel (Section 3.6). We do not build a texture atlas per-frame (alike [Dou et al. 2017]), nor we update it over time (alike [Orts-Escolano et al. 2016]), but rather we apply standard video compression to stream the 8 RGB views to a client; the known camera calibrations are then used to color the rendered model via *projective-texturing*.

3.1 Panoptic 4D capture and reconstruction

We built a capture setup with ×8 RGBD cameras placed around the performer capable of capturing a panoptic view of the scene; see Figure 3. Each of these cameras consists of ×1 RBG camera, ×2 IR cameras that have been mutually calibrated, and an uncalibrated IR pattern projector. We leverage state-of-the-art disparity estimation system by Kowdle et al. [2018] to compute depth very efficiently. The depth maps coming from different viewpoints are then merged following modern extensions [Dou et al. 2017] of traditional fusion paradigms [Dou et al. 2016; Innmann et al. 2016; Newcombe et al. 2015] that aggregate observations from all cameras into a single representation.

Implicit geometry representation. An SDF is an implicit representation of the 3D volume and it can be expressed as a function $\phi(\mathbf{z}) : \mathbb{R}^3 \rightarrow \mathbb{R}$ that encodes a surface by storing at each grid point

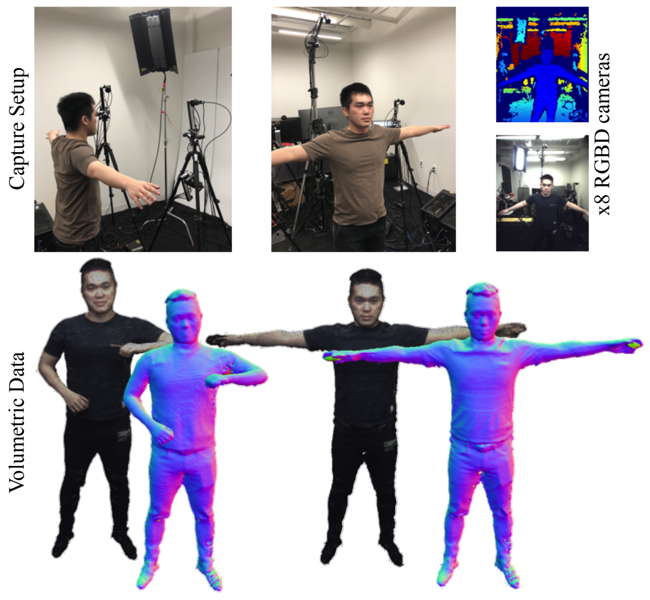

Fig. 3. Our panoptic capture rig, examples of the RGB and depth views and resulting projective texture mapped model.

the signed distance from the surface $\mathcal{S}$ – positive if outside, and negative otherwise. The SDF is typically represented as a dense uniform 3D grid of signed distance values with resolution $W \times H \times D$. A triangular mesh representation of the the surface $\mathcal{S} = \{\mathbf{z} \in \mathbb{R}^3 | \phi(\mathbf{z}) = 0\}$ can be easily extracted via iso-surfacing [de Araujo et al. 2015]; practical and efficient implementations of these methods currently exist on mobile consumer devices. In practical applications, where resources such as computate power and memory are scarce, a Truncated SDF is employed: an SDF variant that is only defined in the neighborhood of the surface corresponding to the SDF zero crossing $\Phi(\mathbf{z}) : \{\mathbf{z} \in \mathbb{R}^3 | \phi(\mathbf{z}) < \varepsilon\} \rightarrow \mathbb{R}$. Although surfaces are most commonly represented explicitly using a triangular/quad mesh, implicit representations make dealing with topology changes easier.

Using our reimplementation of [Dou et al. 2017] we transform the input stream into a sequence of TSDFs $\{\Phi_t\}$ and corresponding sets of RGB images $\{C^n\}_t$ for each frame t.

3.2 Signal compression – Preliminaries

Before diving into the actual 3D volume compression algorithm, we briefly describe some preliminary tools that are used in the rest of the pipeline, namely KLT and range encoding.

3.2.1 Karhunen-Loéve Transform. Given a collection $\{\mathbf{x}_m \in \mathbb{R}^n\}$ of M training signals, we can learn a transformation matrix $\mathbf{P}$ and offset μ that define a forward *KLT* transform, as well as its inverse *iKLT*, of an input signal $\mathbf{x}$ (a transform based on PCA decomposition):

$$\mathbf{X} = \text{KLT}(\mathbf{x}) = \mathbf{P}^T(\mathbf{x} - \mu) \qquad \text{(forward transform)} \qquad (1)$$

$$\mathbf{x} = \text{iKLT}(\mathbf{X}) = \mathbf{P}\mathbf{X} + \mu \qquad \text{(inverse transform)} \qquad (2)$$

Given $\mu = \mathbb{E}_m[\mathbf{x}_m]$, the matrix $\mathbf{P}$ is computed via eigendecomposition of the covariance matrix $\mathbf{C} = \mathbb{E}_m\left[(\mathbf{x}_m - \mu)^T(\mathbf{x}_m - \mu)\right]$ resulting in the matrix factorization $\mathbf{C} = \mathbf{P}\,\Sigma\,\mathbf{P}^T$. Where Σ is a diagonal matrix whose entries $\{\sigma_n\}$ are the eigenvalues of $\mathbf{C}$, and the columns of $\mathbf{P}$ are the corresponding eigenvectors. In particular, if the signal to be compressed is drawn from a linear manifold of dimensionality $d \ll n$, the trailing $(n - d)$ entries of $\mathbf{X}$ will be zero. Consequently, we can encode a $\mathbf{x} \in \mathbb{R}^n$ signal in a lossless way by only storing its $\mathbf{X} \in \mathbb{R}^d$ transform coefficients. Note that to invert the process, the decoder *only* necessitates the transformation matrix $\mathbf{P}$ and the mean vector μ. In practical applications, our signals are not strictly drawn from a low-dimensional linear manifold, but the entries of the diagonal matrix Σ represents how the energy of the signal is distributed in the latent space. In more details, we could only use the first d basis vectors to encode/decode the signal, more formally:

$$\hat{\mathbf{X}} = \text{KLT}_d(\mathbf{x}) = (\mathbf{P}\mathbf{I}_d)^T(\mathbf{x} - \mu) \qquad \text{(lossy encoder)} \qquad (3)$$

$$\hat{\mathbf{x}} = \text{iKLT}_d(\mathbf{X}) = \mathbf{P}\mathbf{I}_d\hat{\mathbf{X}} + \mu \qquad \text{(lossy decoder)} \qquad (4)$$

Where $\mathbf{I}_d$ is a diagonal matrix where only the first d elements are set to one. The reconstruction error is bound by:

$$\epsilon(d) = \mathbb{E}_m\left[\|\mathbf{x}_m - \hat{\mathbf{x}}_m\|_2^2\right] = \sum_{i=d+1}^{n} \sigma_i \qquad (5)$$

For linear latent spaces of order d, not only does the *optimality theorem* of the KLT basis ensure that the error above is zero (i.e. lossless compression), but that the transformed signal $\hat{\mathbf{X}}$ has minimal entropy [Jorgensen and Song 2007]. In the context of compression, this is significant, as entropy captures the average number of bits necessary to encode the signal. Furthermore, similar to the Fourier Transform (FT), as the covariance matrix is symmetric, the matrix $\mathbf{P}$ is orthonormal. Uncorrelated bases ensure that adding more dimensions always adds more information, and this is particularly useful when dealing with variable bitrate streaming. Note that, in contrast to compression based on the FT, the KLT_d compression bases are *signal-dependent*: we train $\mathbf{P}$ and then stream $\mathbf{P}_d$, containing the first d columns of $\mathbf{P}$ to the client. Note this operation could be done offline as part of a codec installation process.

In practice, the full covariance matrix of an entire frame with millions of voxels would be extremely large, and populating it accurately would require a dataset several orders of magnitude larger than ours. Further, dropping eigenvectors would also have a detrimental global smoothing effect. Besides, a TSDF is a sparse signal with interesting values centered around the implicit surface only, which is why we apply KLT to individual, non-overlapping *blocks* of size 8×8×8, which is large enough to contain interesting local surface characteristics, and small enough to capture its variation using our dataset. More specifically, in our implementation we use $5mm$ voxel resolution, so a block has a volume of $40 \times 40 \times 40mm^3$. The learned eigenbases from these blocks need to be transmitted only once to the decoder.

3.2.2 Range Encoding. The d coefficients obtained through KLT projection can be further compressed using a range encoder, which is a lossless compression technique specialized for compressing entire sequences rather than individual symbols as in the case of Huffman coding. A range encoder computes the cumulative distribution function (CDF) from the probabilities of a set of discrete symbols, and recursively subdivides the range of a fixed precision integer respectively to map an entire sequence of symbols into an integer [Martin 1979]. However, to apply range encoding to a sequence of real valued descriptors as in our case, one needs to quantize the symbols first. As such, despite range encoding being a lossless compression method on discrete symbols, the combination results in lossy compression due to quantization error. In particular, we treat each of the d components as a separate message, as treating them jointly may not lend itself to efficient compression. This is due to that fact that the distribution of symbols in each latent dimension can be vastly different and that the components are uncorrelated because of the KLT. Hence, we compute the CDF, quantize the values, and encode the sequence separately for each latent dimension.

3.3 Offline Training of KLT / Quantization

The tools in the previous section can be applied directly to transform a signal, such as an SDF, into a bit stream. The ability, however, to actually compress the signal greatly depends on how exactly they are applied. In particular, applying these tools to the entire signal (e.g. the full TSDF) does not expose repetitive structures that would enable effective compression. As such, inspired by image compression techniques, we look at 8×8×8 sized blocks where we

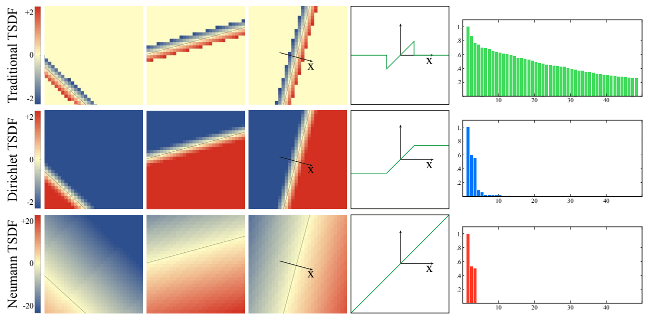

Fig. 4. (left) A selection of random exemplar for several variants of TSDF training signals that are used to train the KLT. (middle) A cross section of the function along the segment **x**, highlighting the different levels of continuity in the training signal. (right) When trained on 10000 randomly generated 32 × 32 images, the KLT spectra is heavily affected by the smoothness in the training dataset.

are likely to see repetitive structure (e.g. planar structures). Further, it turns out that leveraging a training set to choose an effective set of KLT parameters (i.e.**P** and μ) is considerably nuanced, and one of our contributions is showing below how to do that effectively. Finally we demonstrate how to use this training set to form an effective quantization scheme.

Learning **P**, μ *from 2D data.* The core of our compression architecture exploits the repetitive *geometric* structure seen in typical SDFs. For example, the walls of an empty room if axis aligned can largely be explained by a small number of basis vectors that represent axis aligned blocks. When non-axis aligned planar surfaces are introduced, one can reconstruct an arbitrary block containing such a surface by linearly blending a basis of blocks containing planes at a sparse sampling of orientations. Even for surfaces, the SDF is an identity function along the surface normal indicating that the amount of variation in an SDF is much lower than for arbitrary 3D function. We thus use KLT to find a latent space where the variance on each component is known, which we will later exploit in order to encourage bits to be assigned to components of high variance. As we will see, however, applying this logic naively to generate a training set of blocks sampled from a TSDF can have disastrous effects. Specifically, one must carefully consider how to deal with blocks that contain unobserved voxels; the block pixels colored in black in Figure 2. This is complicated by the fact that although real world objects have a well defined inside and outside, it is not obvious how to discern this for the *unobserved voxels* outside the truncation band. The three choices that we consider are:

Traditional TSDF Assigning some arbitrary constant value to to all such unassigned voxel. This will create sharp discontinuities at the edge of the truncation band; i.e. presumably the representation used by Canelhas et al. [2017].

Dirichlet TSDF Only consider "unambiguous" blocks where using the values of voxels on the truncation boundary (i.e. with a value nearly ±1) to flood-fill any unobserved voxels does not introduce large C^0 discontinuities.

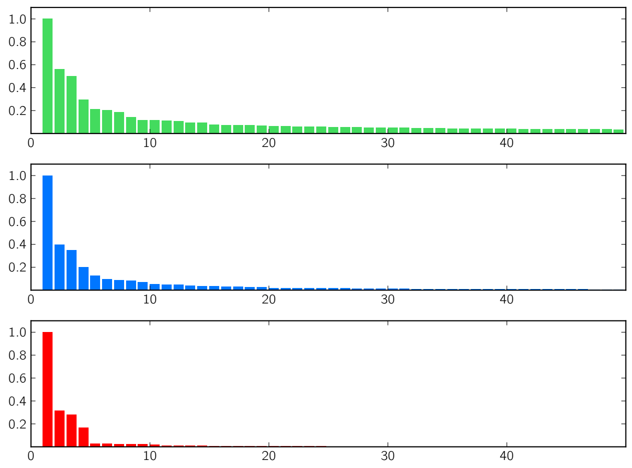

Fig. 5. The (sorted) eigenvalues (diagonal of Σ) for a training signal derived from a collection of 4D sequences. Note the decay rate is slower than that of Figure 4, as these eigenbases need to represent surface details, and not just the orientation of a planar patch.

Neumann TSDF Increase the truncation band and consider blocks that *never* contain an unobserved value. These blocks will look indistinguishable from SDF blocks at training time, but requires additional care at test time when the truncation band is necessarily reduced for performance reasons; see Equation 9.

To elucidate the benefits of these increasingly complicated approaches, we use an illustrative 2D example that naturally generalizes to 3D; see Figure 4. In particular, we consider training a basis on 32×32 blocks extracted from a 2D TSDF containing a single planar surface (i.e. a line). Notice that for the truly SDF like data resulting from the Neumann strategy, the eigenvalues of the eigendecomposition of our signal quickly falls to zero, faithfully allowing the zero-crossing to be reconstructed with a small number of basis vectors. In contrast, the continuous but kinked signals resulting from the Dirichlet strategy extend the spectra of the signal as higher frequency components are required to reconstruct the sharp transition to the constant valued plateaus. Finally, the Traditional strategy massively reduces the compactness of the spectra as the decomposition tries to model the sharp discontinuities in the signal. In more details, to capture 99% of the energy in the signal, Traditional TSDFs require $d = 94$ KLT basis vectors, Dirichlet TSDFs require 14 KLT basis vectors, and Neumann TSDFs require $d = 3$ KLT basis vectors. Note how Neumann is able to discover the true dimensionality of the input data (i.e. $d = 3$).

Learning **P**, μ *from 3D captured data.* Extending the example above to 3D, we train our KLT encoder on a collection of 3D spatial blocks extracted from 4D sequences we acquired using the capture system described in Section 3.1. To deal with unobserved voxels, we apply the Neumann strategy at training time and carefully design our test time encoder to deal with unobserved voxels that occur with the smaller truncation band required to run at real time; see Equation 9. Note that these *training sequences are excluded* in the quantitative evaluations of Section 4. In contrast to the blocks extracted by the *dicer* module at test time, the training blocks need not be non-overlapping, so instead we extract them by sliding-window over every training frame $\{\Phi_m\}$. The result of this operation is summarized by the KLT spectra in Figure 5. As detailed in Section 4, the number d of employed basis vectors is chosen according to the desired tradeoff between bitrate and compression quality.

Learning the quantization scheme. To be able to employ symbolic encoders, we need to develop a quantization function $Q(\cdot)$ that maps the latent coefficients $\hat{\mathbf{X}}$ to a vector of *quantized* coefficients $\hat{\mathbf{Q}}$ as $\hat{\mathbf{Q}} = Q(\hat{\mathbf{X}})$. To this end, we quantize each dimension h of the latent space individually by choosing K_h quantization bins[1] $\{\eta_k^h\}_{k=1}^{K_h} \subseteq \mathbb{R}$ and defining the h'th component of the quantization function as

$$Q_h(\hat{\mathbf{X}}) = \underset{k \in \{1,2,\dots K_h\}}{\arg\min} (\hat{\mathbf{X}}^h - \eta_k^h)^2 . \tag{6}$$

To ensure there are bins near coefficients that are likely to occur we minimize

$$\underset{\{w_{k,n}^h\},\{\eta_k^h\}}{\arg\min} \sum_{k=1}^{K_h} \sum_{n=1}^{N} w_{k,n}^h (\hat{\mathbf{X}}_n^h - \eta_k^h)^2, \tag{7}$$

where $\hat{\mathbf{X}}_n^h$ is the h'th coefficient derived from the n'th training example in our training set. Each point has a boolean vector affiliating it to a bin $\mathcal{W}_n^h = [w_{1,n}^h, \dots, w_{K_h,n}^h]^T \in \mathbb{B}^{K_h}$ where only a single value is set to 1 such that $\sum_i w_{i,n}^h = 1$. This is a 1-dimensional form of the classic K-means problem and can be solved in closed form using dynamic programming [Grønlund et al. 2017]. The eigenvalues corresponding to the principal components of the data are a measure of variance in the projected space. We leverage this by assigning a varied number of bins to each component based on their respective eigenvalues, which range between $[0, V_{\text{total}}]$. In particular, we adopt a linear mapping between the standard deviation of the data and number of bins, resulting in $K_h{=}K_{\text{total}}\sqrt{V_h}/\sqrt{V_{\text{total}}}$ bins for eigenbasis h. The components receiving a single bin are not encoded since their values are fixed for all the samples, and they are simply added as a bias to the reconstructed result in the decompression stage. As such, the choice of K_{total} partially controls the trade off between the reconstruction accuracy and bitrate.

3.4 Run-time 3D volume compression/decompression

Encoder. At run time, the *encoding architecture* entails the following steps: A variant of *KLT*, designed to deal with unobserved voxels, maps each of the $m = 1 \dots M$ blocks to a vector of coefficients $\{\hat{\mathbf{X}}_m\}$ in a low-dimensional latent space (Section 3.4.2). We compute histograms on coefficients of each block, and then *quantize* them into the set $\{\hat{\mathbf{Q}}_m\}$; these histograms are re-used to compute the optimal quantization thresholds (Section 3.4.3). To further reduce bitrate allocations, the lossless *range encoder* module compresses the symbols into a bitstream.

Decoder. At run time, the *decoding architecture* mostly inverts the process described above, although due to quantization the signal can only be *approximately* reconstructed; see Section 3.4.4. The decode process produces a set of coefficients $\{\hat{\mathbf{X}}_m\}$. However, at low bitrates, this results in blocky-artifacts typical of JPEG compression. Since we know that the compressed signal is a manifold, these artifacts can be removed by means of an optimization designed to remove C^1 discontinuities from the signal (Section 3.4.5). Finally, we extract the zero crossing of Φ via a GPU implementation of [Lorensen and Cline 1987]. This extracts a triangular mesh representation of $\mathcal{S}$ that can be texture mapped.

We now detail all the above steps, which can run in real-time even on mobile platforms.

3.4.1 Dicer. As we stream the content of a TSDF only the observed blocks are transmitted. To reconstruct the volume we therefore need to encode and transmit the block contents as well as the block indices. The blocks are transmitted with indices sorted in an ascending manner and we use delta encoding to convert the vector of indices $[i, j, k, \dots]$ into a vector $[i, j-i, k-j, \dots]$. This conversion makes the vector entropy encoder friendly as it becomes a set of small duplicated integers. Unlike the block data, we want lossless reconstruction of the indices so we cannot train the range encoder beforehand. This means that for each frame we need to calculate the CDF and the code book to compress the index vector.

3.4.2 KLT – compression. The full volume Φ is decomposed into a set $\{\mathbf{x}_m\}$ of non-overlapping 8×8×8 blocks. In this stage, we seek to leverage the learnt basis $\mathbf{P}$ and μ to map each $\mathbf{x} \in \mathbb{R}^n$ to a manifold with much lower dimensionality d. We again, however, need to deal with the possibility of blocks containing unobserved voxels. At training time, it was possible to apply the Neumann strategy. At test time, however, we cannot increase the truncation region as it would reduce the frame rate of our non-rigid reconstruction pipeline. To this end, we instead minimize the objective:

$$\underset{\hat{\mathbf{X}}}{\arg\min} \|\mathbf{W}[\mathbf{x} - (\mathbf{P}\mathbf{I}_d\hat{\mathbf{X}} + \mu)]\|. \tag{8}$$

The matrix $\mathbf{W} \in \mathbb{R}^{n\times n}$ is a diagonal matrix where the i-th diagonal entry is set to 1 for observed voxels, and 0 otherwise. The least-squares solution of the above optimization problem can be computed in closed form as:

$$\hat{\mathbf{X}} = (\mathbf{I}_d^T\mathbf{P}^T\mathbf{W}\mathbf{P}\mathbf{I}_d)^{-1}[(\mathbf{P}\mathbf{I}_d)^T\mathbf{W}(\mathbf{x} - \mu)]. \tag{9}$$

Note that when all the blocks are observed this simplifies to Equation 3. Computing $\hat{\mathbf{X}}$ requires the solution of a small $d \times d$ dimensional linear problem, that can be efficiently computed.

3.4.3 Quantization and encoding. We now apply our quantization scheme to the input coefficients $\{\hat{\mathbf{X}}_m\}$ via $\hat{\mathbf{Q}}_m = Q(\hat{\mathbf{X}}_m)$ to obtain the set of quantized coefficients $\{\hat{\mathbf{Q}}_m\}$. For each dimension h, we apply the range encoder to those coefficients $\{\hat{\mathbf{Q}}_m^h\}$, across all blocks, as if they were symbols so as to losslessly compress them, leveraging any low entropy empirical distributions that appears over these symbols. The bitstream from the range encoder is sent directly to the client.

3.4.4 iKLT – decompression. The decompressed signal can be efficiently obtained via Eq. 4, namely: $\hat{\mathbf{x}} = \mathbf{P}\mathbf{I}_d\hat{\mathbf{X}} + \mu$. If enough bins were used during quantization, this will result in a highly accurate, but possibly high bitrate, reconstruction.

[1]Note that what we refer to as "bins" are really just real numbers that implicitly define a set of intervals on the real line through (6).

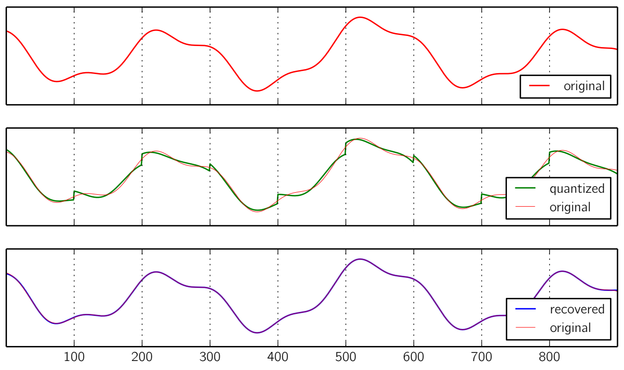

Fig. 6. (top) A smooth signal is to be compressed by representing each non-overlapping block of 100 samples. (middle) By compressing with KLT$_4$ (or 99% of the variance explained), the signal contains discontinuities at block boundaries. (bottom) Our variational optimization recovers the quantized KLT coefficients, and generates a C^1 continuous signal.

3.4.5 iQuantizer – block artifacts removal. Due to our block compression strategy, even when using a number d of KLT basis vectors that explain 99% of the variance in the training data, the decompressed signal presents C^0 discontinuities at block boundaries; see Figure 6 for a 2D example. If not enough bins are used during quantization, these can become quite pronounced. To address this problem, we try to use a set of corrections $\tilde{\mathbf{X}}$ to the quantized coefficients and look at the corrected reconstruction

$$\tilde{\mathbf{x}}_m(\tilde{\mathbf{X}}) = \left[\mathbf{P}\mathbf{I}_d(\hat{\mathbf{Q}} + \tilde{\mathbf{X}}_m)\right] + \mu. \tag{10}$$

We know that the corrections are in the intervals defined by the quantization coefficients. That is

$$\tilde{\mathbf{X}}_m \in C = [-\epsilon_m^1, \epsilon_m^1] \times \ldots \times [-\epsilon_m^d, \epsilon_m^d]\ . \tag{11}$$

where for dimension $h \in \{1, \ldots d\}$, $\epsilon_m^h \in \mathbb{R}$ is half the width of the interval that (6) assigns to bin center $\hat{\mathbf{Q}}_m^h$. To find the corrections, we penalize the presence of discontinuities *across blocks* in the reconstructed signal. We first abstract the *iDicer* functional block by parameterizing the reconstructed TSDF in the entire volume $\Phi(\mathbf{z}) = \tilde{\mathbf{x}}(\mathbf{z}|\{\tilde{\mathbf{X}}_m\}) : \mathbb{R}^3 \to \mathbb{R}$. We can then recover the corrections $\{\tilde{\mathbf{X}}_m\}$ via a variational optimization that penalizes discontinuities in the reconstructed signal $\tilde{\mathbf{x}}$:

$$\underset{\{\tilde{\mathbf{X}}_m\}}{\arg\min} \sum_{i,j,k} \|\Delta\Phi_{ijk}(\mathbf{z})(\{\tilde{\mathbf{X}}_m\})\|^2 \text{ where } \{\tilde{\mathbf{X}}\} \subseteq C. \tag{12}$$

Here Δ is the Laplacian operator of a volume, and ijk indexes a pixel in the volume. In practice, the constraints in (11) are hard to deal with, so we relax the constraint as

$$\underset{\{\tilde{\mathbf{X}}_m\}}{\arg\min} \sum_{i,j,k} \|\Delta\Phi_{ijk}(\mathbf{z})(\{\tilde{\mathbf{X}}_m\})\|^2 + \lambda \sum_m \sum_{h=1}^{d} \left(\frac{\tilde{\mathbf{X}}_m^h}{\epsilon_m^h}\right)^2, \tag{13}$$

a linear least squares problem where λ is a weight indicating how strongly the relaxed constraint is enforced. See Figure 2 for a 2D illustration of this process, and Figure 7 to see its effects on the rendered mesh model.

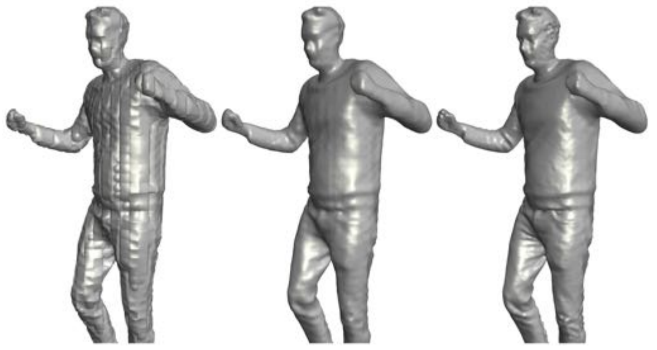

Fig. 7. (left) A low-bitrate TSDF is received by the decoder and visualized as a mesh triangulating its zero crossing. In order to exaggerate the block-artifacts for visualization, we choose $K_{Total} = 512$ in this example. (middle) Our filtering technique removes these structured artifacts via variational optimization. (right) The ground truth triangulated TSDF.

3.5 Color Compression

Although the scope and contribution of this paper is geometry compression, we also provide a viable solution to deal with texturing. Since we are targeting real-time applications, building a consistent texture atlas would break this requirement [Collet et al. 2015]. We recall that in our system we use only 8 RGB views, therefore we decided to employ traditional video encoding algorithms to compress all the images and stream them to the client. The receiver, will collect those images and apply projective texturing method to render the final reconstructed mesh.

In particular, we employ a separate thread to process the N streams from the RGB cameras via the real-time H.264 GPU encoder (NVIDIA NVENC). This can be efficiently done on an Nvidia Quadro P6000, which has two dedicated chips to off-load the video encoding, therefore not interfering with the rest of the reconstruction pipeline. In terms of run time, it takes around 5 ms to compress all of the data when both the encoding chips present on the GPU are used in parallel. The same run time is needed on the client side to decode. The bandwidth required to achieve a level of quality with negligible loss is around 8 Mbps. Note that the RGB streams coming from the N RGB cameras are *already* temporally consistent, enabling standard H.264 encoding.

3.6 Networking and live demo setup

Our live demonstration system consists of two main parts, connected by a low speed link: a high performance backend collection of capture/fusion workstations (×10 HP Z840), a WiFi 802.11a router (Netgear AC1900), and a remote display laptop (Razer Blade Pro). The backend system divides the capture load into a hub and spoke model. Cameras are grouped into pods, each consisting of two IR cameras for depth and one color camera for texture; each camera is operating at 1280×1024 @ 30Hz. As detailed in Section 3.1, each pod also has an active IR dot pattern emitter to support active stereo.

In our demo system, there are eight pods divided among the capture workstations. Each capture workstation performs synchronization, rectification, color processing, active stereo matching, and

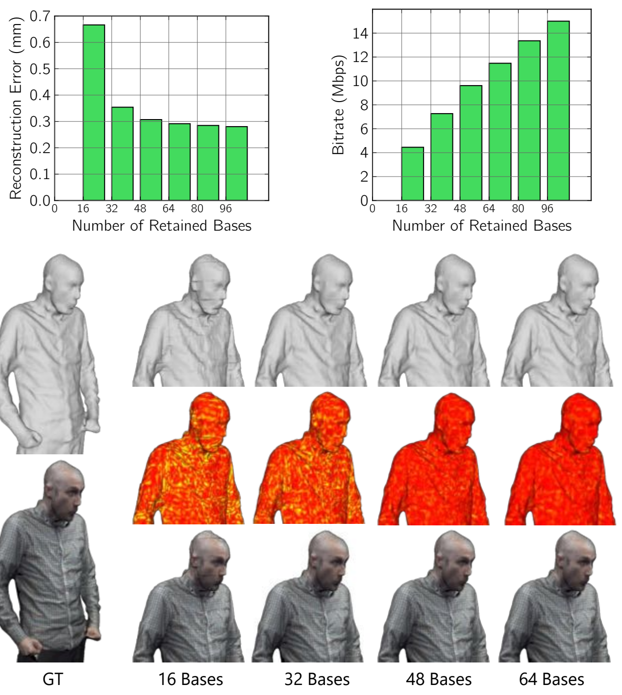

Fig. 8. Measuring the impact of the number of employed KLT basis vectors on (top-left) reconstruction quality and (top-right) bitrate. (bottom) Qualitative examples when retaining different number of basis vectors. Note we keep $K_{total} = 2048$ fixed in this experiment.

segmentation. Each capture workstation is directly connected to a central fusion workstation using 40 GigE links. Capture workstations transmit segmented color (24 bit RGB) and depth (16 bit) images using run-length compression to skip over discarded regions of the view over TCP using a request/serve loop. On average, this requires a fraction of the worst-case of 1572 Mbps required per pod, around 380 Mbps. A "fusion box" workstation combines the eight depth maps and color images into a single textured mesh for local previewing. To support real-time streaming of the volume and remote viewing, the fusion box also compresses the TSDF volume representation and the eight color textures. The compressed representation, using the process described in Section 3.4 for the mesh and hardware H.264 encoding for the combined textures is computed for every fused frame at 30 hertz. On average each compressed frame requires about 20 KB. The client laptop, connected via Wi-Fi, uses a request/serve loop to receive the compressed data and performs decompression and display locally.

4 EVALUATION

In this section, we perform quantitative and qualitative analysis of our algorithm, and compare it to our re-implementation of state-of-the-art techniques. We urge the reader to watch the **accompanying video** where further comparisons are provided, as well as a live recording of our demo streaming a 4D performance in *real-time* via WiFi to a consumer-level laptop. Note that in our experiments we focus our analysis around two core aspects:

- the requirement for real-time (30 fps in both encoder/decoder) performance
- the compression of geometry (i.e. not color/texture)

4.1 Datasets and metrics

We acquire three RGBD datasets by capturing the performance of three different subjects at 30Hz. Each sequence, ≈500 frames in length, was recorded with the setup described in Section 3.6. To train the KLT basis of Section 3.2 we randomly selected 50 random frames from *Sequence 1*. We then use this basis to test on the remaining ≈1450 frames.

We generate high quality reconstructions that are then used as ground-truth meshes. To do so we attempted to re-implement the FVV pipeline of Collet et al. [2015]. We first aggregate the depth maps into a shared coordinate frame, and then register the multiple surfaces to each other via local optimization [Bouaziz et al. 2013] to generate a merged (oriented) point cloud. We then generate a watertight surface by solving a Poisson problem [Kazhdan and Hoppe 2013], where screening constraints enforce that parts of the volume be outside as expressed by the visual hull. We regard these frames as ground truth reconstructions. Note that this process does not run in real-time, with an average execution budget of 2 min/frame. We use it *only* for quantitative evaluations, and it is not employed in our demo system.

Typically to decide the degree to which two geometric representations are identical, the Hausdorff distance is employed [Cignoni et al. 1998]. However, as a single outlier might bias our metric, in our experiments we employ an ℓ_1 relaxation of the Hausdorff metric [Tkach et al. 2016, Eq. 15]. Given our decompressed model X, and a ground truth mesh Y, our metric is the (average, and area weighted) ℓ_1 medians of (mutual) closest point distances:

$$\mathcal{H}(X,Y) = \frac{1}{2A(X)} \sum_{x \in X} A(x) \inf_{y \in Y} d(x,y) + \frac{1}{2A(Y)} \sum_{y \in Y} A(y) \inf_{x \in X} d(x,y)$$

where x is a vertex in the mesh X, and $A(.)$ returns the area of a vertex, or of the entire mesh according to its argument.

4.2 Impact of compression parameters

In our algorithm, reconstruction error, as measured by $\mathcal{H}$, can be attributed to two different parameters: (Section 4.2.1) the number of learned KLT basis vectors d; and (Section 4.2.2) the number of quantization bins K_{total}.

4.2.1 Impact of KLT dimensionality – Figure 8. We show the impact of reducing the number d of basis vectors on the reconstruction error as well as on the bitrate. Although the bitrate increases almost linearly in d, the reconstruction error plateaus for $d > 48$. In the qualitative examples, when $d = 16$, blocky artifacts remain even after our global solve attempts to remove them. When $d = 32$, the reconstruction error is still high, but the final global solve manages to remove most of the blocky artifacts, and projective texturing makes the results visually acceptable. When $d \geq 48$ it is difficult to notice any differences from the ground truth.

4.2.2 Impact of quantization – Figure 9. We now investigate the impact of quantization when fixing the number of retained basis vectors. As shown in the figure, in contrast to d, the bitrate changes

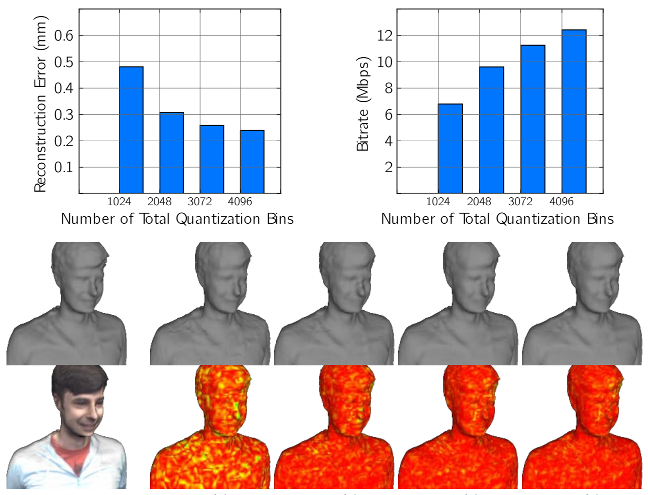

Fig. 9. Measuring the impact of quantization on (top-left) reconstruction quality and (top-right) bitrate. (bottom) Qualitative examples when employing a different number of quantization bins. Note we keep $d = 48$ fixed in this experiment.

Table 1. Average timings for encoding a single of 300k faces (175k vertices) across different methods. On the encoder side, our compressor (with $K_{Total} = 2048$, $d = 48$) is orders of magnitude faster than existing techniques. Since our implementation of FVV [Collet et al. 2015] is not efficient, we report the encoding speed from their paper when ≈ 70 workstations were used – note no decoding framerate was reported.

Method	Encoding	Decoding
Ours (CPU / GPU)	17 ms / **2 ms**	19 ms / **3ms**
Draco (qp=8 / 14)	180 ms / 183 ms	81 ms / 98 ms
FVV	25 seconds	N/A
JP3D	14 seconds	25 seconds

sub-linearly with the number of total assigned bins K_{Total}. When $K_{Total} = 2048$, the compressor produces a satisfying visual output with a bandwidth comparable to high-definition (HD) broadcast.

4.3 Comparisons to the state-of-the-art – Figure 10

We compare with two state-of-the-art geometry compression algorithms: *FVV* [Collet et al. 2015] and *Draco* [Galligan et al. 2018]. Whilst Draco is tailored for quasi-lossless static mesh compression, FVV is designed to compress 4D acquired data similar to the scenario we are targeting. Qualitative comparisons are reported in Figure 11, as well as in the **supplementary video**.

The Google Draco [Galligan et al. 2018] mesh compression library implements a modern variant of EdgeBreaker [Rossignac 1999], where a triangle spanning tree is built by traversing vertices one at a time – as such, this architecture is difficult to parallelize, and therefore not well-suited for real-time applications; see Table 1. Draco uses the parameter qp to control level of location quantization, i.e., the trade-off between compression rate and loss. With our dataset, we observe artifacts by vertices collapsing when $qp \leq 8$, hence we only vary qb between 8 and 14 (recommended setting). The computational time increases altogether with this parameter. At the same level of reconstruction error, their output file size is 8x larger than ours; see Figure 10 (left).

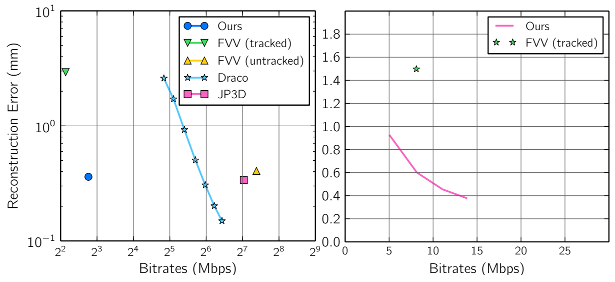

Fig. 10. (left) Comparison with prior arts on our dataset. Note that since some results are orders of magnitude apart, *semilogy* axes are used. To generate a curve Draco [Galligan et al. 2018], we vary the parameter quantization bits (qp). (right) Comparison with FVV [Collet et al. 2015] on their BREAKDANCERS sequence. FVV's bitrate was obtained from their paper (by removing the declared texture bitrate). Our rate-distortion curve has been generated by varying the number of KLT basis d.

As detailed in Section 4, we re-implemented Free Viewpoint Video [Collet et al. 2015] in order to generate high quality, uncompressed reconstructions and compare their method on our acquired dataset. The untracked, per-frame reconstructions are used as ground-truth meshes. The method in Collet et al. [2015] tracks meshes over time and performs an adaptive decimation to preserve the details on the subject's face. We compare our system with FVV tracked, where both tracking and decimation are employed, and with FVV untracked, where we only use the smart decimation and avoid the expensive non-rigid tracking of the meshes.

Our method, outperforms FVV (tracked) in terms of reconstruction and bitrate. The FVV (untracked), but decimated, generally has lower error but prohibitive bitrates (around 128 Mbps). Figure 10 shows that our method ($K_{Total} = 2048$, $d = 32$) achieves similar error with the untracked version of FVV with ≈ 24x lower bitrate. Notice that whereas FVV focuses on exploiting dynamics in the scene to achieve compelling compression performances, our methods mostly focus on encoding and transmitting the keyframes, therefore we could potentially combine the two to achieve better compression rates. Finally, it is important to highlight again that FVV is much slower than our method. In Collet et al. [2015], they describe the tracked version has a speed of ≈ 30 minutes per frame on a single workstation.

We also compare our approach with JP3D [Schelkens et al. 2006], an extension of the JPEG2000 codec to volumetric data. Similar to our approach, JP3D decomposes volumetric data as blocks and processes them independently. However, while our algorithm uses KLT as a data driven transformation, JP3D uses a generic discrete wavelet transform to process the blocks. The wavelet coefficients are then quantized before being compressed with an arithmetic encoder. In the experiment, we use the implementation in OpenJPEG, where 3 successive layers are chosen with compression ratio 20, 10, and 5. The other parameters are kept as default. Since JP3D assumes a dense 3D volume as input, even though it does not use expensive temporal tracking, encoding time of each frame still takes 14 seconds while decoding takes 25 seconds. With similar reconstruction error, their output bitrate is 131Mbps, which is still one order of magnitude larger than ours.

5 LIMITATIONS AND FUTURE WORK

We have introduced the *first* compression architecture for dynamic geometry capable of reaching real-time encoding/decoding performance. Our fundamental contribution is to pivot away from triangular meshes towards implicit representations, and, in particular, the TSDFs commonly used for non-rigid tracking. Thanks to the simple memory layout of a TSDF, the volume can be decomposed into equally-sized blocks, and encoding/decoding efficiently performed on each of them independently, in parallel.

In contrast to [Collet et al. 2015] that utilizes temporal information, our work focuses on encoding an entire TSDF volume. Nonetheless, the compression of post motion-compensation residuals truly lies at the foundation of the excellent compression performance of modern video encoders [Richardson 2011]. An interesting direction for future works would be how to leverage the $SE(3)$ cues provided by motion2fusion [Dou et al. 2017] to encode post motion-compensation residuals. While designed for efficiency, our KLT compression scheme is linear, and not particularly suited to represent piecewise $SE(3)$ transformations of rest-pose geometry. An interesting venue for future works would be the investigation of *non-linear* latent embeddings that could produce more compact, and hence easier to compress, encodings. Despite their evaluation in real-time remaining a challenge, convolutional auto-encoders seem to be a perfect choice for the task at hand [Rippel and Bourdev 2017].

REFERENCES

Sofien Bouaziz, Andrea Tagliasacchi, and Mark Pauly. 2013. Sparse Iterative Closest Point. *Computer Graphics Forum (Symposium on Geometry Processing)* (2013).

Stéphane Calderon and Tamy Boubekeur. 2017. Bounding proxies for shape approximation. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 57.

Daniel-Ricao Canelhas, Erik Schaffernicht, Todor Stoyanov, Achim J Lilienthal, and Andrew J Davison. 2017. Compressed Voxel-Based Mapping Using Unsupervised Learning. *Robotics* (2017).

Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. 1998. Metro: Measuring error on simplified surfaces. (1998).

David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. 2004. Variational shape approximation. *ACM Trans. on Graphics (Proc. of SIGGRAPH)* (2004).

Alvaro Collet, Ming Chuang, Pat Sweeney, Don Gillett, Dennis Evseev, David Calabrese, Hugues Hoppe, Adam Kirk, and Steve Sullivan. 2015. High-quality streamable free-viewpoint video. *ACM Trans. on Graphics (TOG)* (2015).

Brian Curless and Marc Levoy. 1996. A volumetric method for building complex models from range images. In *SIGGRAPH*.

B. R. de Araujo, Daniel S. Lopes, Pauline Jepp, Joaquim A. Jorge, and Brian Wyvill. 2015. A Survey on Implicit Surface Polygonization. *Comput. Surveys* (2015).

Mingsong Dou, Philip Davidson, Sean Ryan Fanello, Sameh Khamis, Adarsh Kowdle, Christoph Rhemann, Vladimir Tankovich, and Shahram Izadi. 2017. Motion2fusion: real-time volumetric performance capture. *ACM Trans. on Graphics (Proc. of SIGGRAPH Asia)* (2017).

Mingsong Dou, Sameh Khamis, Yury Degtyarev, Philip Davidson, Sean Ryan Fanello, Adarsh Kowdle, Sergio Orts Escolano, Christoph Rhemann, David Kim, Jonathan Taylor, et al. 2016. Fusion4d: Real-time performance capture of challenging scenes. *ACM Transactions on Graphics (TOG)* (2016).

Frank Galligan, Michael Hemmer, Ondrej Stava, Fan Zhang, and Jamieson Brettle. 2018. Google/Draco: a library for compressing and decompressing 3D geometric meshes and point clouds. https://github.com/google/draco. (2018).

Michael Garland and Paul S Heckbert. 1997. Surface simplification using quadric error metrics. In *Proc. of ACM SIGGRAPH*.

Allan Grønlund, Kasper Green Larsen, Alexander Mathiasen, Jesper Sindahl Nielsen, Stefan Schneider, and Mingzhou Song. 2017. Fast exact k-means, k-medians and bregman divergence clustering in 1d. *arXiv preprint arXiv:1701.07204* (2017).

Hugues Hoppe. 1996. Progressive Meshes. In *Proc. of ACM SIGGRAPH*.

Matthias Innmann, Michael Zollhöfer, Matthias Nießner, Christian Theobalt, and Marc Stamminger. 2016. VolumeDeform: Real-time volumetric non-rigid reconstruction. In *Proc. of the European Conf. on Comp. Vision*. 362–379.

Palle ET Jorgensen and Myung-Sin Song. 2007. Entropy encoding, Hilbert space, and Karhunen-Loeve transforms. *J. Math. Phys.* 48, 10 (2007), 103503.

Zachi Karni and Craig Gotsman. 2000. Spectral compression of mesh geometry. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co.

Zachi Karni and Craig Gotsman. 2004. Compression of soft-body animation sequences. *Computers & Graphics* (2004).

Michael Kazhdan and Hugues Hoppe. 2013. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)* (2013).

Adarsh Kowdle, Christoph Rhemann, Sean Fanello, Andrea Tagliasacchi, Jonathan Taylor, Philip Davidson, Mingsong Dou, Kaiwen Guo, Cem Keskin, Sameh Khamis, David Kim, Danhang Tang, Vladimir Tankovich, Julien Valentin, and Shahram Izadi. 2018. The Need 4 Speed in Real-Time Dense Visual Tracking. (2018).

Bruno Lévy and Hao (Richard) Zhang. 2010. Spectral Mesh Processing. In *ACM SIGGRAPH Courses*.

William E Lorensen and Harvey E Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *Computer Graphics (Proc. SIGGRAPH)*, Vol. 21. 163–169.

Adrien Maglo, Guillaume Lavoué, Florent Dupont, and Céline Hudelot. 2015. 3d mesh compression: Survey, comparisons, and emerging trends. *Comput. Surveys* (2015).

G Nigel N Martin. 1979. Range encoding: an algorithm for removing redundancy from a digitised message. In *Proc. IERE Video & Data Recording Conf., 1979*.

MPEG4/AFX. 2008. *ISO/IEC 14496-16: MPEG-4 Part 16, Animation Framework eXtension (AFX)*. Technical Report. The Moving Picture Experts Group. https://mpeg.chiariglione.org/standards/mpeg-4/animation-framework-extension-afx

Richard A Newcombe, Dieter Fox, and Steven M Seitz. 2015. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proc. of Comp. Vision and Pattern Recognition (CVPR)*.

Sergio Orts-Escolano, Christoph Rhemann, Sean Fanello, Wayne Chang, Adarsh Kowdle, Yury Degtyarev, David Kim, Philip L Davidson, Sameh Khamis, Mingsong Dou, et al. 2016. Holoportation: Virtual 3d teleportation in real-time. In *Proc. of the Symposium on User Interface Software and Technology*.

Jingliang Peng, Chang-Su Kim, and C-C Jay Kuo. 2005. Technologies for 3D mesh compression: A survey. *Journal of Visual Communication and Image Representation* (2005).

Fabián Prada, Misha Kazhdan, Ming Chuang, Alvaro Collet, and Hugues Hoppe. 2017. Spatiotemporal atlas parameterization for evolving meshes. *ACM Trans. on Graphics (TOG)* (2017).

Iain E Richardson. 2011. *The H. 264 advanced video compression standard*. John Wiley & Sons.

Oren Rippel and Lubomir Bourdev. 2017. Real-time adaptive image compression. *arXiv preprint arXiv:1705.05823* (2017).

Jarek Rossignac. 1999. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics* (1999).

Michael Ruhnke, Liefeng Bo, Dieter Fox, and Wolfram Burgard. 2013. Compact RGBD Surface Models Based on Sparse Coding. (2013).

Peter Schelkens, Adrian Munteanu, Joeri Barbarien, Mihnea Galca, Xavier Giro-Nieto, and Jan Cornelis. 2003. Wavelet coding of volumetric medical datasets. *IEEE Transactions on medical Imaging* (2003).

Peter Schelkens, Adrian Munteanu, Alexis Tzannes, and Christopher M. Brislawn. 2006. JPEG2000 Part 10 - Volumetric data encoding. (2006).

Olga Sorkine, Daniel Cohen-Or, and Sivan Toledo. 2003. High-Pass Quantization for Mesh Encoding.. In *Proc. of the Symposium on Geometry Processing*.

Robert W Sumner, Johannes Schmid, and Mark Pauly. 2007. Embedded manipulation for shape manipulation. *ACM Trans. on Graphics (TOG)* (2007).

Jean-Marc Thiery, Emilie Guy, and Tamy Boubekeur. 2013. Sphere-Meshes: Shape Approximation using Spherical Quadric Error Metrics. *ACM Trans. on Graphics (Proc. of SIGGRAPH Asia)* (2013).

Jean-Marc Thiery, Émilie Guy, Tamy Boubekeur, and Elmar Eisemann. 2016. Animated Mesh Approximation With Sphere-Meshes. *ACM Trans. on Graphics (TOG)* (2016).

Anastasia Tkach, Mark Pauly, and Andrea Tagliasacchi. 2016. Sphere-Meshes for Real-Time Hand Modeling and Tracking. *ACM Transaction on Graphics (Proc. SIGGRAPH Asia)* (2016).

Sébastien Valette and Rémy Prost. 2004. Wavelet-based progressive compression scheme for triangle meshes: Wavemesh. *IEEE Transactions on Visualization and Computer Graphics* 10, 2 (2004), 123–129.

Libor Vasa and Vaclav Skala. 2011. A perception correlated comparison method for dynamic meshes. *IEEE transactions on visualization and computer graphics* (2011).

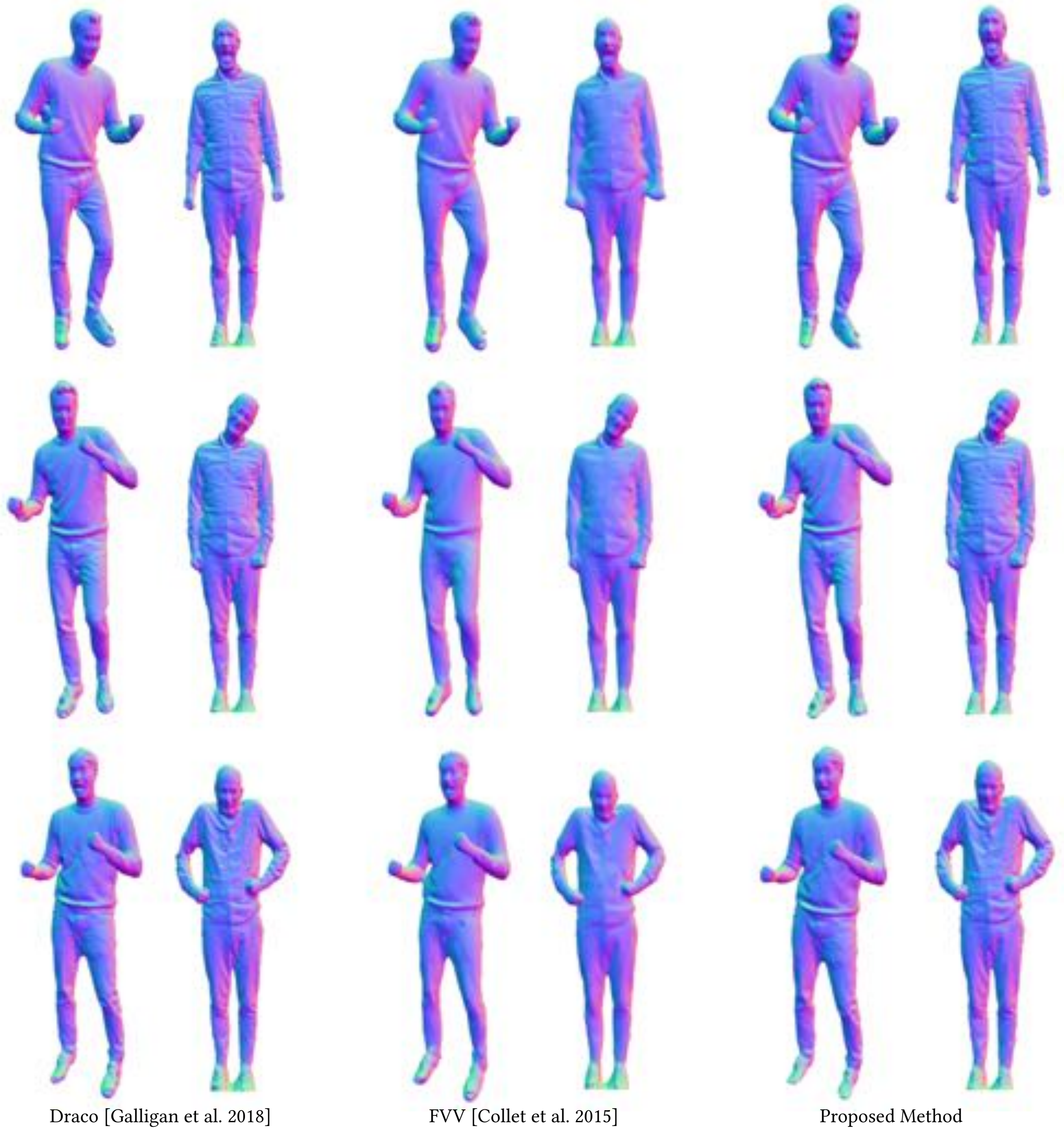

Fig. 11. Qualitative comparisons on three test sequences. We compare our approach with Draco [Galligan et al. 2018] (qp=14) and FVV [Collet et al. 2015]. Notice how FVV over-smooths details especially in the body regions. Draco requires bitrates that are 10 times larger than ours with results that are comparable to ours in terms of visual quality.