Benchmarking Open-Source Static 3D Mesh Codecs for Immersive Media Interactive Live Streaming

Alexandros Doumanoglou*, Petros Drakoulis*, Nikolaos Zioulis, Dimitrios Zarpalas, and Petros Daras, *Senior Member, IEEE*

Abstract—This work provides a systematic understanding of the requirements of live 3D mesh coding, targeting (tele-)immersive media streaming applications. We thoroughly benchmark in rate-distortion and runtime performance terms, four static 3D mesh coding solutions that are openly available. Apart from mesh geometry and connectivity, our analysis includes experiments for compressing vertex normals and attributes, something scarcely found in literature. Additionally, we provide a theoretical model of the tele-immersion pipeline that calculates its expected frame-rate, as well as lower and upper bounds for its end-to-end latency. In order to obtain these measures, the theoretical model takes into account the compression performance of the codecs and some indicative network conditions. Based on the results we obtained through our codec benchmarking, we used our theoretical model to calculate and provide concrete measures for these tele-immersion pipeline's metrics and discuss on the optimal codec choice depending on the network setup. This offers deep insight into the available solutions and paves the way for future research.

Index Terms—3D Compression, Mesh Coding, Tele-Immersion, 3D Media Streaming, Performance Evaluation, Time Varying Meshes.

1 Introduction & Related Work

We have reached a milestone where it is now possible to capture high quality 3D content in real-time. When remotely transmitted, this new type of three-dimensional (3D) media can facilitate the concepts of tele-presence [1], [2] and tele-immersion (TI) [3], or otherwise more recently referred to as Holoportation [4]. Contemporary TI platforms [2], [3], [4] produce 3D media content in the form of Time-Varying Meshes (TVMs), which - in contrast to dynamic meshes - are challenging to compress efficiently in an online manner. This stems from their varying vertex and triangle counts, and therefore connectivity, across frames.

Consequently, TVM compression is accomplished either by using specialized TVM codecs that exploit inter-frame redundancy or by compressing each frame individually, using standard static 3D mesh coding. A noteable example for the former is [5] which, however, has increased time complexity and cannot operate at high frame rates in order to support real-time applications. For the latter, most works focus on encoding time-varying geometry and neglect connectivity. In [6] a first attempt to efficiently encode TVM geometry is discussed. However, this method does not deal with connectivity coding and also requires storing an increased amount of side-information in order to properly decode the geometry of the TVM stream. In [7] the authors improve the geometry coding, but otherwise handle connectivity in an equally inefficient way.

The ongoing MPEG-I [8] standardization work also focuses on point cloud coding [9], which deals with time-varying geometry omitting the connectivity information. Two notable recent works focusing on point cloud compression are [10] and [11] which deal with motion estimation using octree subdivided macro-blocks and spectral graph transforms respectively, albeit operating at rates prohibitive for remote interaction scenarios. Moreover, point cloud representations require specialized rendering and very dense sampling in order to reach the levels of fidelity that textured meshes provide.

All these characteristics, namely the lack of proper connectivity handling and the incapability of current TVM inter-frame codecs to operate at real-time speeds, along with the relatively immature state of current point cloud solutions, make the use of static mesh codecs for real-time TI streaming a quite appealing choice.

In this work, we focus on benchmarking the performance of existing, open-source, static 3D mesh compression methods, in the context of 3D immersive media interactive live streaming[1]. More specifically, we consider the following codecs: Google Draco [12], Corto [13], MPEG's Open 3D Graphics Compression (O3dgc) [14] and OpenCTM [15]. Among those open-source implementations, Draco is based on [16], Corto is based on [17], O3dgc is based on [18], while OpenCTM is the only library which is not based on any academic publication.

While summarizing surveys on 3D mesh compression present in the literature [19], [20], [21], currently, there is a gap on the evaluation of contemporary 3D mesh codecs

* *Indicates equal contribution.*

- *A. Doumanoglou, P. Drakoulis, N. Zioulis, D. Zarpalas and P. Daras are with the Visual Computing Lab (VCL) of the Information Technologies Institute (ITI), Centre for Research and Technology Hellas (CERTH), Thessaloniki, Greece.*
  E-mail: {aldoum, petros.drakoulis, nzioulis, zarpalas, daras}@iti.gr.
  Website: vcl.iti.gr.

1. We consider live streaming in the context of interactive (tele-) immersion media applications and not live broadcasting scenarios. The latter has relaxed latency requirements, but the former imposes more strict latency restrictions to enable multi-user interactions.

in a consistent experimental setup that takes into account not only geometric rate-distortion (RD) performance of the codecs but also their RD performance when compressing normals and attributes, as well as their time complexity, which is crucial for live streaming. This paper aims to be one of the first works towards filling this scarcity with its main contributions being:

- A thorough and extensive benchmarking, based on more than 50000 experiments of the most popular available open source static 3D mesh codecs in a consistent experimental setup, taking into account their RD performance not only for vertex geometry but also for normals and attributes.
- A run-time performance benchmarking of these codecs which uncovers their time complexity.
- A live TI streaming case study which estimates their impact on the end-to-end performance of the TI pipeline, in terms of frame-rate and latency.

2 Evaluated 3D Mesh Codecs Overview

In this section, a brief overview of the underlying encoding algorithms that are utilized by the studied open-source 3D mesh codecs is presented. All codecs use a similar geometry quantization scheme that is based on quantizing the mesh coordinates to a predefined grid, sized according to the meshes' bounding box. For Draco, Corto and O3dgc we give details on their connectivity compression algorithm while for OpenCTM, apart from the connectivity compression algorithm we give further details on its geometry compression scheme which slightly deviates from the rest of the codecs.

2.1 Corto

Corto is a simple, yet fast, mesh compression algorithm. The encoding process visits one triangle at a time and maintains a list of the edges of the processed region's boundary. The processed region is always homeomorphic to a disk and is always growing, as new neighboring triangles to the region are being visited. The three edges of the first triangle are first added to the list. Then, iteratively one edge is being extracted from the list and the algorithm encodes the relation of the not-yet-visited triangle incident to the edge with respect to the boundary of the already encoded region. Four cases are distinguished:

- **SKIP:** This is the case when the extracted edge is a boundary edge (i.e. it has no other non-processed triangles adjacent) or the adjacent triangle is already processed.
- **LEFT or RIGHT:** These cases denote the condition when the adjacent to the extracted edge triangle, shares two edges with the processed region's boundary.
- **VERTEX:** This case indicates that the adjacent to the extracted edge triangle, shares exactly one edge with the processed region's boundary.

Fig. 1 depicts all the previously mentioned cases.

2.2 Draco

Draco uses Edgebreaker [16] as its underlying mesh compression algorithm. Edgebreaker traverses the 3D mesh in a series of steps. At each step the algorithm visits and encodes one not-yet-visited triangle of the mesh. At each stage the input mesh is divided into disjointed regions that may share a vertex but no edges. The edges bounding each region constitute a polygonal curve which is called a "loop". The edges of the loop are called "gates" with one gate being active at each step. At every step, there is a triangle incident to the active gate that is not yet visited. Let v denote the vertex of this triangle that is not incident to the gate. Edgebreaker encodes the relation of v with respect to the gate's loop boundary and the gate itself. Edgebreaker stands out 5 cases labeled: C, L, E, R, S. When v does not belong to the active gate's loop, this case is marked as C. When v belongs to the active gate's loop and is also incident to the active gate, then this condition is marked as R, L or E, depending on the side of the active gate where v is incident to (R or L). When v is incident to both sides of the active gate the condition is marked as E. Finally, the condition where v belongs to the loop but is not incident to the active gate, is marked as S. In Fig. 2 an example of all Edgebreaker cases is depicted.

2.3 O3dgc

O3dgc uses TFAN [18] as its mesh compression algorithm. TFAN is based on traversing the mesh vertices from neighbor to neighbor. At each step of the process, one vertex is marked as the focus vertex. TFAN traverses the input mesh by visiting triangles incident to the focus vertex in the order they appear in the Triangle Fan (TF) representation. In the TF representation, an ordered set of vertices $\{v_0, v_1, v_2, ..., v_{d+1}\}$ form a set of d triangles $\{t_j\}$ such that $\forall j \in \{0, 1, ..., d-1\}, t_j = \{v_0, v_{j+1}, v_{j+2}\}$. In that case, v_0 is considered to be the center of the TF. At each step of TFAN, a new focus vertex is extracted from a queue and the set of its incident triangles is partitioned in TFs. Let $O(v_j) = j$ denote the traversal order of the j-th vertex extracted from the queue, i.e. $O(v_j)$ denotes the order in which v_j was extracted from the queue. Let also $L(j)$ denote the ordered set of vertices sharing with v_j at least one visited triangle. The vertices in the set $L(j)$ are ordered by their traversal order. All the vertices of the TF of the focus vertex are traversed is the order they define the TF. Let "active" vertex denote the currently traversed vertex of the TF. If the active vertex of the TF has been previously visited, a binary value of 0 is emitted to the S binary vector, while 1 is emitted in the opposite case. When the active vertex is visited for the first time, it is marked as visited and is inserted to the queue in order to later process it as a focus vertex. Further, the active vertex is inserted to the set $L(j)$. In the case where the active vertex was already visited, two different conditions are considered. In the first case, the active vertex is already included to the set $L(j)$. In that case, the relative index of the vertex inside $L(j)$ is stored into an additional integer vector I. On the other hand, if the active vertex does not belong to $L(j)$, the traversal order difference between the active vertex of the TF and the focus vertex, which consists a negative value to aid decoding, is stored in I.

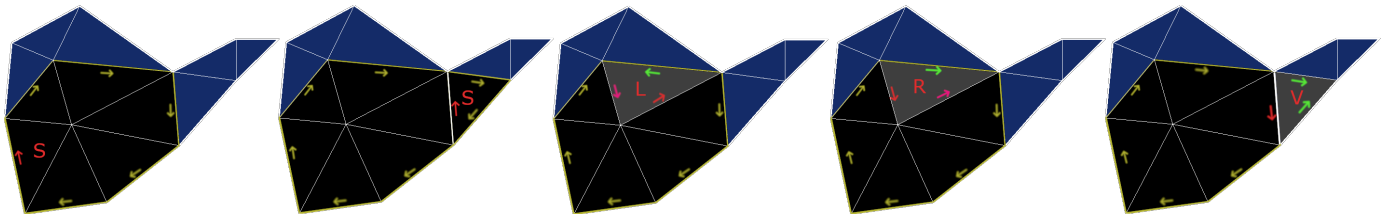

Fig. 1. Enumerating Corto various cases. From left to right: (S)kip, (S)kip, (L)eft, (R)ight, (V)ertex. Already processed triangles in black. Not-yet-processed triangles in blue. Currently encoded triangle in gray. Current active edge in red. Boundary edges in yellow and removed boundary edges in magenta. Newly added edges in green.

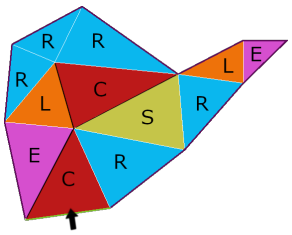

Fig. 2. Edgebreaker triangle traversal example. The active gate where traversal begins is in green. The grown boundary as the triangle traversing evolves in time in black.

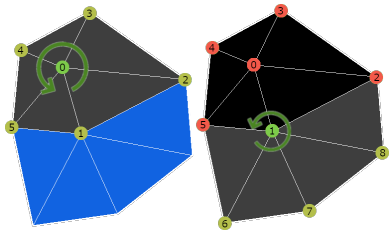

Fig. 3. Example of TFAN coding. Already processed triangles in black. Not-yet-processed triangles in blue. Currently encoded TF in gray. Focus vertex in green. Vertices visited for the first time are in yellow. Vertices already visited in red. Vertices are labeled according to their traversal order as focus vertices. Left: Encoding vectors: $S = \{1, 1, 1, 1, 1\}, I = \{\}$. Right: $S = \{0, 1, 1, 1, 0\}, I = \{2, 1\}$.

This procedure continues until all mesh vertices have been processed as focus vertices. TFAN, further improves upon compression ratio by exploiting common configurations of the S and I vectors and encoding them as a single integer in the C vector and only further encoding parts of S and I as appropriate.

2.4 OpenCTM

OpenCTM has three different algorithm variants namely: Raw, MG1 and MG2. In this study, we consider only the MG2 variant since it is the one performing always the best in rate-distortion terms. MG2 is a simple algorithm that is based on sorting and delta coding for both geometry and connectivity. Initially, a grid that fits the bounding box of the mesh is constructed and the mesh vertices are sorted based on their index inside the grid and subsequently (upon equality) based on their x coordinate. The difference between each vertex and its grid cell's origin is computed and quantized based on the requested precision. Then, the sorted vertices' grid indices are delta coded and appended to the output bitstream along with the quantized delta coordinates. For connectivity, the standard 3 integer representation is considered, with each integer referencing one vertex of each triangle. Firstly, the vertex references inside each triangle are rearranged so that the first index is the smallest one. Then, the triangles are sorted based on their first and second indices, with the second indices being used when the first indices are equal. Finally, a delta coding scheme on the final triangle list is applied so that the first index of each triangle is delta coded with respect to the previous triangle in the list, while the other two indices of the triangle are encoded as a difference with respect to the first index of the same triangle.

3 Tele-immersive Streaming

Tele-immersion is based on the next-generation video, which is 3D video of live performances and is grounded on two pillars: i) real-time 3D reconstruction of dynamic scenes and ii) real-time compression and transmission of the generated 3D media. There are two main directions that can be pursued for the production of 3D media. The first one reconstructs the user out of acquired sensor data in a per frame basis and is also the direction that preliminary attempts in capturing humans in full 3D initially pursued [1], [2], [22]. Recently, another direction emerged as non-rigid registration in real-time became possible. These recent non-rigid production systems [4] fuse all information into a canonical model representation by deforming the input on a per frame basis, achieving higher fidelity results as a consequence of the implicit denoising that integrating temporal information into the canonical model offers. Despite their differences however, both types of 3D production methods extract an isosurface from an implicit representation and transform it into a triangulated surface, using the marching cubes algorithm [23]. This results in a TVM $\mathbb{M} : \{\mathbf{V}, \mathbf{T}\}$, where $\mathbf{V}$ and $\mathbf{T}$ are the set of vertices and triangle indices respectively.

This geometric representation facilitates an elevated sense of presence as it can naturally position a user's 3D representation into a shared space, be it either virtual or real, in appropriate scale and in a coherent manner with respect to the surroundings. However, fully transferring a user's identity in 3D requires photorealism which is achieved by also capturing and transmitting her/his realistic appearance in the form of color information. The aforementioned 3D capture and production systems [3], [4] accomplish this by additionally transmitting the color camera feeds in real-time, which are then used as supplementary textures to the geometry. In order to render the final representation in a

photorealistic and high quality manner, these textures need to be blended appropriately during rendering. This means that additional information that will drive the blending of the textures needs to be transmitted. Such metadata typically involve blending weights $\mathbf{b}$ and/or texture identifiers $\mathbf{i}$. From a 3D codec point of view, they are mainly distinguished by their data type, be it either floating or fixed point data. Furthermore, realistic embedding of the transmitted 3D representation also requires lightening the renderings appropriately, so as to appear blended into the environment. This requirement is met with the transmission of extra surface information in the form of the surface's normals $\mathbf{n}$.

We can thus conclude that streaming TI payloads consist of texture and geometry $\mathbb{M} : \{\mathbf{V}, \mathbf{T}, \mathbf{N}, \mathbf{A}\}$ encoded information, where $\mathbf{N}$ and $\mathbf{A} : \{\mathbf{B}, \mathbf{I}\}$ are the set of normals and attributes accompanying the TVM, with the attributes comprising of the necessary metadata (blending weights and texture indices respectively) to render the TVM. As a result, 3D codecs used for encoding TI TVMs need to also encode a variety of custom attributes in addition to the vertex $\mathbf{V}$ and connectivity $\mathbf{T}$ information. Therefore, when comparing different codecs and analyzing their performance, we need to follow a holistic approach and benchmark heterogeneous payload coding efficiency, as well as computational performance, given that the targeted application requires minimal latency to facilitate multi-party interactions.

4 Codec Evaluation

In this section, we discuss on the benchmarking of the studied 3D mesh codecs with respect to all different evaluation aspects. First, in subsection 4.1, we begin by commenting on the generation of the TI dataset which all codecs are evaluated on.

Most of the available codecs are able to be configured in ways that allow faster encoding / decoding times at the cost of achieving lower compression rates for the same distortion level. An exhaustive benchmarking of all available configurations of each codec is not practical. Thus, for each codec, we have chosen some preset values for those configurations that span the available parameter ranges. We call those preset configurations codec "*profiles*". The specifications of those profiles that we have chosen to benchmark are given in detail in subsection 4.2.

Subsequently, in subsection 4.3, we report RD performance of the codecs for three different cases: a) encoding connectivity along with vertex positions (also referred to as "geometry"), b) encoding connectivity along with vertex positions and normals and c) encoding connectivity along with vertex positions and attributes. In their respective subsections, we also comment on the methodology and the employed error metrics. Not all profiles affect the rate-distortion performance in all of the previously mentioned cases (i.e. there are profiles that only affect the compression of normals and not positions, or the compression of attributes and not normals or positions, etc.). Thus, only relevant profiles are being benchmarked for each case, while for illustration purposes, equivalent profiles are grouped together.

Finally, in subsection 4.4, we report the relative performance for all codec profiles while also adding another dimension into the analysis, runtime performance. To that end, we benchmark the time taken to compress and decompress the TI meshes. This is accomplished by setting target distortion values for all attributes specific to the TI streaming scenario, which are used to drive the relative performance analysis. This analysis is conducted for the three aforementioned cases, as well as for the complete (i.e. full) case that includes vertex positions, normals and attributes all together.

4.1 Dataset

It is apparent, that the RD performance of any codec does not solely depend on the chosen profile, but also on the structure of the provided input mesh. The most relevant 3D mesh codec evaluations that are presented in the literature, evaluate the codecs' performance on 3D meshes generated by 3D artists, or meshes that are generated by a surface reconstruction algorithm applied on high precision range scanned data. The meshes generated by either of those methods are clean and free of noise, as opposed to 3D meshes that are generated by 3D reconstruction methods that are applied on data acquired by depth sensors operating at high frame-rates.

In this work, we focus our analysis on a practical application of mesh compression, live TI streaming. Consequently, we use a set of meshes generated by an actual TI pipeline, and more specifically by the 3D reconstruction method of [22]. Our TI dataset comprises 10 randomly selected distinct frames (i.e. 3D models) depicting 5 different subjects with variability in poses, as the subjects were captured while executing different performances (i.e. punching, kicking, conversing, expressing emotions, etc.). We use 10 different models with their average vertex count being 16868, spanning the range of $[10242 - 21064]$ vertices, and their average triangle count being 30713, spanning the range of $[18792 - 36520]$ triangles.

4.2 Codec Profiles

In this subsection, we give a brief overview of the available configuration options that are made accessible via the programmable interface of each 3D mesh codec participating in this benchmark. For each one of them, the values of the corresponding configuration implicitly affects its rate-distortion performance and execution time. As previously introduced in the beginning of Section 4, we select and name specific "profiles" that comprise sets of specific configuration options for each codec. In the following subsections these profiles and their options are presented.

4.2.1 Corto

Corto encodes geometry and vertex normals at a quality specified in the form of quantization bits while custom floating point vertex attributes are quantized using an explicit quantization step. Custom integer vertex attributes are not directly supported. To overcome this limitation, we treat these attributes as floating points quantized with a quantization step of 1.0 unit. Corto is benchmarked in two variations that differ only in the normals prediction scheme. In Corto's

default profile, which we call "*Corto1*", the normals are estimated from the quantized geometry and their differences with the quantized actual normals is encoded using the octahedron projection representation [24]. In Corto's second profile ("*Corto2*") the quantized normals in the octahedron projection representation are delta coded with respect to a neighboring quantized normal belonging to a quad incident to the normal's vertex. In both profiles, the encoding of the normals are driven by the Corto's connectivity traversal of the mesh.

4.2.2 Draco

Draco encodes all kinds of floating point vertex attributes (positions, normals and custom attributes) at a detail level specified by a given number of quantization bits. Custom integer attributes are supported as well. In our case, the per-vertex texture identifiers are losslessly encoded as integers, while the texture blending weights' precision is controlled by a specified number of quantization bits. The Draco interface allows adjusting the processing time versus compression ratio mixture via a "speed" setting expressed on an integer scale from 10 (denoting the combination of options that minimize processing time) to 0 (denoting the ones that lead to the most possibly compressed representation). We chose to benchmark three configurations, namely "*Draco2*", "*Draco5*" and "*Draco9*", with the speed setting set to 2, 5 (the default) and 9 (the fastest setting possible that utilizes the underlying EdgeBreaker [16] algorithm) respectively.

4.2.3 O3dgc

Apart from standard support for controlling geometric loss via an externally provided number of quantization bits, and in a way similar to Draco, O3dgc also supports vertex normals as well as custom integer and floating point vertex attributes. Using a combination of available options, we opted for the benchmark of three encoding configurations that differ in the geometry and custom attributes' prediction strategy. The "default" profile (*O3d*) makes use of parallelogram prediction [25] for the geometry, differential prediction for the custom floating point attributes and no prediction for the integer ones. The "fast" profile (*O3f*) uses differential prediction for the geometry and no prediction for both integer and floating point custom attributes. Finally, the "small" profile (*O3s*) uses differential prediction for all, geometry, integer and floating point vertex attributes. In all of the aforementioned selected profiles, the (unit) normals are first converted into a representation that exploits unit sphere inscription in a cube. Apart from parallelogram prediction which only applies to geometry, in the differential prediction mode each vertex position, normal or attribute is delta coded with respect to the value of the corresponding position, normal or attribute belonging to a neighboring vertex based on the connectivity of the mesh.

4.2.4 OpenCTM

Instead of specifying the number of quantization bits, OpenCTM only allows controlling the geometric, normal and attribute loss via an explicit quantization step. OpenCTM does not support integer attributes and thus, as with Corto, we encode the per-vertex texture identifiers in a floating point representation using a quantization step of 1.0 unit. Apart from the standard OpenCTM implementation which uses LZMA (a variant of [26]) for entropy compression, we have implemented a custom version which replaces the LZMA entropy compression module with LZ4 [27] (a faster but less efficient variant of [26]). Therefore, we have chosen to use two OpenCTM profiles, called "*CTM-LZMA*" and "*CTM-LZ4*", which both use the OpenCTM's internal MG2 algorithm but differ in the lossless entropy compression algorithm. Both entropy compression algorithms were chosen to be operated at the fastest possible speed setting, favoring fast execution times over high compression ratio.

4.3 Rate-Distortion Performance Evaluation

In this subsection we report RD curves for the cases (a), (b) and (c) that were introduced in the beginning of this section. For each case we report average performance in bit-rate and the corresponding distortion metric across all the models of the respective dataset. This is accomplished via averaging the bit-rate and the distortion metric for constant quantization step or number of quantization bits across the different models of the dataset. This analysis leads to extracting average codec performance in real-world captured data for each fixed value of the parameters that control distortion.

4.3.1 Case A: Connectivity & Vertex Positions

First, we evaluate the codecs' average RD performance when compressing geometry and connectivity only. The geometric distortion metric that we use is the standard Hausdorff distance between the compressed and the uncompressed meshes which is reported by the METRO [28] tool with respect to (wrt) the mesh's bounding box diagonal. For the evaluation of the distortion metric, the METRO tool was used with its default parameterization to sample vertices, edges and faces by taking a number of samples that is approximately 10 times the number of triangles in the model.

Benchmark results

The benchmark results of this experiment for all profiles of the mesh codecs are depicted in Fig. 4, top. It can be easily observed that in geometry RD terms, both OpenCTM profiles perform significantly worse than the other codecs, hampering graph's readability. Thus, a detailed view that helps to better illustrate the performance differences of the other codecs, in their various profiles, is also offered as an inset.

The best codec in terms of RD is the slowest profile of Draco. The second best profile in the same terms is *Draco5*, which has very similar performance to *O3d*. Third in order, *O3f/s* profiles perform similar to *Draco9*. Excluding OpenCTM, Corto has the least efficient algorithm in terms of RD, producing about one half the compression rates produced by the best profile of Draco, for high to mid distortion levels.

Finally, another important element depicted in Fig. 4 is that, for all codecs, when compressing geometry alone, the geometric distortion of the compressed mesh is solely determined by the value of the quantization parameter and

does not depend in any way on the codec profile settings. On the other hand, bit-rate depends on the combination of the quantization parameter's value and the codec profile setting.

4.3.2 Case B: Connectivity, Vertex Positions & Vertex Normals

Secondly, we conduct an experiment to evaluate the RD performance of the various codec profiles for the case of compressing mesh geometry and connectivity jointly with vertex normals. The nature of this experiment is two-dimensional, as the final bit-rate is affected by two quantization parameters: one for the vertex geometry and one for the vertex normals. Each quantization parameter controls the distortion level in its domain respectively. For an evenhanded approach to benchmarking the various codec profiles, we first pick a fixed level of geometric (Hausdorff) distance and for each codec profile we try to tune the quantization parameter for vertex positions to a proper value that leads to this level of geometric distortion. Subsequently, we variate the normal quantization parameter for each codec profile and draw RD curves that illustrate the normal distortion for a given bit-rate. We repeat the above-mentioned process for two different levels of geometric distortion, resulting into 2 different normal RD curves. Since each codec has its own way to perform quantization, exact geometric distortion equivalence across codecs cannot be achieved. However, our strategy to tune the vertex position quantization values for each codec individually, ensures that when evaluating the normal distortion versus bit-rate, the distortion level of vertex positions across codecs is as close as possible to the preset target level.

For the definition of the normal distortion metric, we follow a Hausdorff-like root mean squared error (RMSE) approach that measures the angle difference between the compressed and original mesh's vertex normals, relying on a vertex correspondence strategy based on proximity. More specifically, let $\mathcal{A}$ and $\mathcal{B}$ denote two meshes with vertices $\mathbf{v} \in \mathcal{A}$ and $\mathbf{u} \in \mathcal{B}$. Let also $\mathbf{n}(\mathbf{v})$ and $\mathbf{n}(\mathbf{u})$ denote the normals of vertex $\mathbf{v}$ and $\mathbf{u}$, respectively. Then we define the following error function that calculates the normal distortion from mesh $\mathcal{A}$ to mesh $\mathcal{B}$:

$$n_{err}(\mathcal{A},\mathcal{B}) = \sqrt{\frac{1}{N}\sum_{\mathbf{v}\in\mathcal{A}}\left[\min_{\mathbf{u}\in\mathcal{N}(\mathbf{v})} \angle(\mathbf{n}(\mathbf{v}),\mathbf{n}(\mathbf{u}))\right]^2} \quad (1)$$

with $N \in \mathbb{N}^*$ denoting the vertex count in $\mathcal{A}$ and $\mathcal{N}(\mathbf{v}) = \{\mathbf{u} \in \mathcal{B} : ||\mathbf{u}-\mathbf{v}|| < r\}$, with r a predefined radius distance. Let $\mathcal{M}_r$ denote a reference (uncompressed) mesh and $\mathcal{M}_c$ denote its compressed version. We define the final normal error distortion metric between $\mathcal{M}_c$ and $\mathcal{M}_r$ to be:

$$e_n(\mathcal{M}_r,\mathcal{M}_c) = \max(n_{err}(\mathcal{M}_r,\mathcal{M}_c), n_{err}(\mathcal{M}_c,\mathcal{M}_r)) \quad (2)$$

For each pair of $\mathcal{M}_c$ and $\mathcal{M}_r$ we set the parameter r equal to their Hausdorff distance.

Benchmark results

The benchmark results of this experiment are presented in Fig. 4 middle and bottom sub-figures, with each one presenting results for a different geometric distortion level. One of the most important observations of this experiment is that the RMS error for normals for the first profile of *Corto1* and both profiles of OpenCTM does not converge to zero, even for the higher bit-rates. Thus, in general, they are significantly outperformed in rate distortion terms by the rest of the codecs.

As with the previous case, when compressing geometry along with normals, the best codec in RD terms is *Draco2*, followed by the same codec's *Draco5* profile. For higher geometric distortion (Hausdorff distance with respect to the bounding box diagonal ≈ 0.0027) third in order comes *Draco9*. Fourth and fifth come the O3dgc's profiles (*O3d* and *O3f/s*) respectively, while last in order is *Corto2*. For lower geometric distortion (Hausdorff distance wrt bounding box diagonal ≈ 0.00065) the two O3dgc's profiles perform closer to *Draco9*.

In contrast to the previous case, in the present scenario and for the Corto codec, the distortion of normals in the compressed mesh is not solely controlled by the quantization parameter but is also affected by the codec's profile. In RD terms, the first profile of Corto is clearly surpassed by the second profile of the same Codec. Furthermore, while for Draco, O3dgc and *Corto2* the geometric distortion has an insignificant impact on the final distortion of the compressed normals, this is not the case for *Corto1* and both profiles of OpenCTM. The latter codec profiles compress normals in a way that their distortion is also affected by the distorted geometry of the compressed mesh, with lower geometric loss improving the distortion of the normals for the same quantization parameters.

4.3.3 Case C: Connectivity, Vertex Positions & Vertex Attributes

Last, we benchmark the RD performance of the various codec profiles for the case of compressing mesh geometry and connectivity along with vertex attributes. Similar to the case with normals (Case B - subsection 4.3.2), the nature of this experiment is also two-dimensional, with two quantization parameters controlling geometric and attribute distortions respectively. We follow the same approach as we did previously, by first picking an appropriate value for the corresponding quantization parameter that leads to a preset level of geometric distortion. Then, we proceed in evaluating attribute distortion versus bit-rate by adjusting the attribute quantization parameter to various values. As already discussed in Section 3 in the TI streaming case, the vertices of the meshes have both integer and floating point attributes. Specifically for the case of [22], they have two integer texture identifiers and one floating point blending weight for each texture pair which are used during rendering. In our experiments, the attributes corresponding to the texture identifiers are encoded losslessly while the only attribute that is subject to distortion is the one corresponding to the blending weight. The final bit-rate though, is affected by both types of vertex attributes.

To measure the attribute distortion between the compressed and the original (uncompressed) meshes we define an error metric which resembles the error metric we used for normal distortion. In particular, if $\mathcal{A}$ and $\mathcal{B}$ denote two meshes with vertices $\mathbf{v} \in \mathcal{A}$ and $\mathbf{u} \in \mathcal{B}$ and $\mathrm{a}(\mathbf{v})$ and $\mathrm{a}(\mathbf{u})$ denote a single floating point attribute of vertices $\mathbf{v}$ and

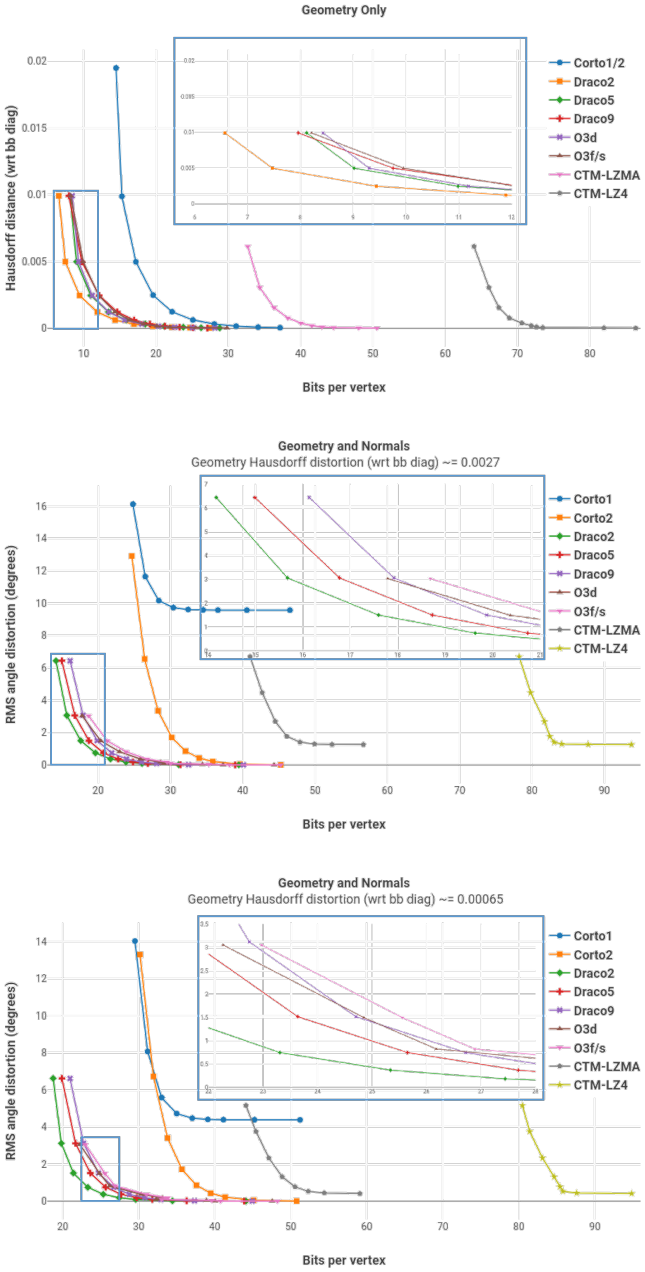

Fig. 4. Rate-distortion curves for all codecs when compressing geometry (vertex positions and connectivity) only - top - and geometry with normals - middle and bottom. For all cases, zoom-ins are provided as insets to better illustrate differences for those codecs whose RD curves are very close. For the normals RD curves, results are shown for a constant geometric distortion level as mentioned in each plot's subtitle.

$\mathbf{u}$, respectively we define the following error function that calculates the attribute distortion from mesh $\mathcal{A}$ to mesh $\mathcal{B}$:

$$a_{err}(\mathcal{A},\mathcal{B}) = \sqrt{\frac{1}{N}\sum_{\mathbf{v}\in\mathcal{A}}\left[\min_{\mathbf{u}\in\mathcal{N}(\mathbf{v})}|\mathrm{a}(\mathbf{v})-\mathrm{a}(\mathbf{u})|\right]^2} \quad (3)$$

with N and $\mathcal{N}(\mathbf{v})$, having the same notion as in subsection 4.3.2. For the uncompressed mesh $\mathcal{M}_r$ and its compressed version $\mathcal{M}_c$, we define the final attribute error distortion metric between $\mathcal{M}_c$ and $\mathcal{M}_r$ to be:

$$e_a(\mathcal{M}_r,\mathcal{M}_c) = \max(a_{err}(\mathcal{M}_r,\mathcal{M}_c), a_{err}(\mathcal{M}_c,\mathcal{M}_r)) \quad (4)$$

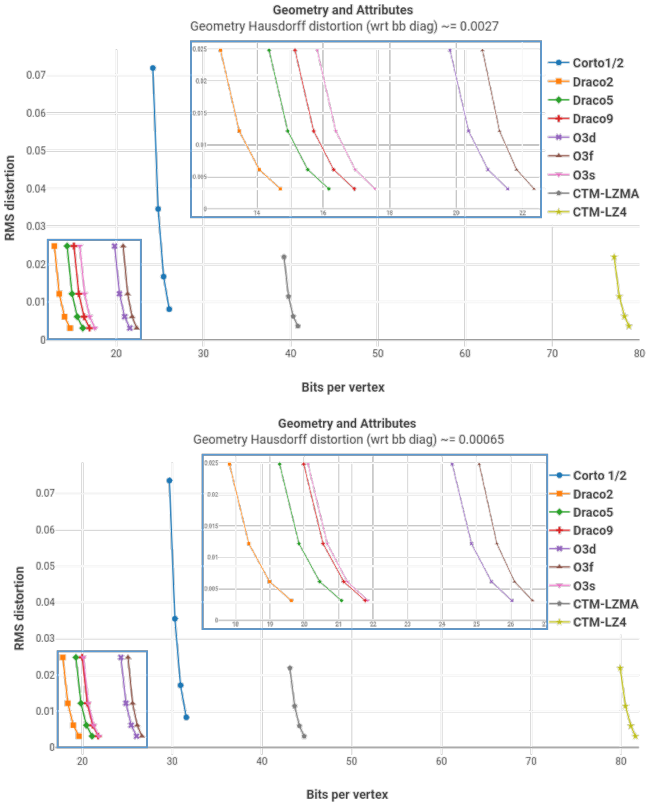

Fig. 5. Rate-Distortion performance (Attribute RMS distortion vs bit-rate, for two fixed geometric distortion levels) when compressing Geometry along with Vertex Attributes. On the top and bottom, rate-distortion comparisons for high and low geometric losses, respectively.

Similar to subsection 4.3.2, for each pair of $\mathcal{M}_c$ and $\mathcal{M}_r$ we set the parameter r equal to their Hausdorff distance.

Benchmark results

We benchmark the various codec profiles for their RD performance in encoding attributes for the two levels of geometric distortion that we set in subsection 4.3.2, namely Hausdorff distance (with respect to the bounding box) $\approx$ 0.0027 and $\approx$ 0.00065. The respective RD curves are depicted in Fig. 5.

Ordering the various codec profiles in RD terms for this particular case is very clear. Draco is the best performing codec under the RD criterion, with its faster profiles consistently producing higher bit-rates for the same level of distortion, but still achieving better RD performance than all other codecs. Fourth in order comes *O3s* followed by *O3d* and *O3f*. Last in order are Corto (both of its profiles perform the same) and the two profiles of OpenCTM.

From Fig. 5 it can be deduced that the final attribute distortion level achieved by all codecs is not affected by their profiles. The codec profiles only affect the final bit-rate. Further, as observed in Fig. 5, the level of geometric distortion has only a small impact on the achieved distortion levels of the attributes. In addition, the ordering of the codecs in RD terms does not change when geometric distortion decreases, except that *O3s* reaches the performance of *Draco9*.

4.4 Relative Comparison of Codec Performances

In the current section of the paper we discuss the relative performance of the studied codec profiles in four aspects of mesh compression namely bit-rate, distortion, encoding and decoding times. We perform the analysis for four total cases: compressing connectivity with vertex positions, compressing connectivity with vertex positions and normals, compressing connectivity with vertex positions and attributes, and finally compressing connectivity along with vertex positions, normals and attributes altogether. An exhaustive comparison in all of those aspects for all the possible values of the quantization parameters is impractical, thus, we aim to set a specific target level of distortion for each generic vertex attribute and compare the various codec profiles in the four aforementioned aspects.

Since our study is motivated by the application of those codecs in a TI scenario, we pick the target distortion levels for geometry, normals and attributes as such, in order to align with this practical use case. In particular, for geometry, we choose the Hausdorff distance with respect to the bounding box diagonal to be ≈ 0.00065 which is equivalent to $\approx$ 0.0003 meters, i.e. 0.3mm RMS error (as reported by the METRO tool) and $\approx$ 0.0012 meters, i.e. 1.2mm of actual Hausdorff distance. For normals, we target the 1° RMS angle distortion and for the attribute corresponding to the texture blending weight (a value which lies in the range $[0-1]$), we aim at an RMS distortion level of about 0.0145. As already discussed in the previous section, the exact target level of distortion, in all the generic vertex attributes, cannot be achieved by all codecs and thus, for each codec profile, we find the value of the quantization parameter that leads to a distortion which is as close as possible to the target level that we have set.

To measure the encoding and decoding times for each codec profile, we average the time taken for each codec to compress / decompress the respective mesh 30 times, in order to estimate the time taken to complete a single compression / decompression pass. For each mesh we convert the time measured by this process in per vertex time units by dividing the time needed for each pass with the mesh's vertex count. Finally, we average the numbers corresponding to the encoding / decoding times per vertex for all the meshes in the dataset to get a representative value of the codecs' runtime performance. At this point it is important to note that all codecs were built from source code with all speed optimization options turned on[2].

The figures that we present show the relative performance of each codec with respect to the performance of the best codec in each aspect and for each case. In all figures, lower values indicate better performance. The figures are in the form of bar charts, with the values of the bars being the average codec performance in the respective aspect, across all models. Additionally, we show error bars that indicate the standard deviation of the measurements around the average for the different models of the dataset.

2. Further, no codec implementation contains any type of assembly optimizations or any CPU specific instruction extension sets.

4.4.1 Case A: Connectivity & Vertex Positions

The results of the first case when compressing connectivity and geometry alone are depicted in Fig. 6. The geometric distortion for all codecs is approximately equal, except for OpenCTM which produced meshes with 26% more geometric distortion than the rest of the codecs. In all aspects and for all codecs, the standard deviation of each metric across the different models of the dataset is relatively small compared to the average value, making the average value a good representative of the codec performances.

Comparing *O3d* with the *O3f/s* profiles, the latter two produce $\approx 4.5\%$ larger bit-rate but are about $\approx 20.5\%$ and $\approx 27.35\%$ faster in encoding and decoding respectively. *Draco9* produces $\approx 18\%$ larger bit rate than *Draco2* but at the same time it is $\approx 20\%$ and $\approx 16.3\%$ faster in encoding and decoding. Thus, Draco's profiles showcase a better trade-off between bit-rate and speed than O3dgc's. Further, comparing Draco with O3dgc, we observe that *Draco2* is better than *O3d* in all aspects while the same holds for *Draco5* when compared to the *O3f/s* profiles. Corto has the faster encoder about $\approx 30.5\%$ faster than *Draco9* but at the same time producing $\approx 48.3\%$ larger bit-rate. The fastest decoder is *CTM-LZ4*, but coming at a very increased bit-rate compared to other codecs. Further, the performance of *CTM-LZMA* is inferior to all profiles of Draco, Corto and the *O3f/s* profiles in all aspects.

In order to showcase the discrepancy in RD between quick and performant encoding we also offer BD-rates [29], [30] for those codecs lying at these opposite performance ends. While Fig. 6 showcases Draco's superiority in producing low bit-rates, there is no clear decision between Corto and OpenCTM with respect to encoding/decoding speed. To that end, we also factor in their relative performance in lower bit-rate production, and thus, offer BD-rate comparisons for Draco and Corto. Table 1 presents the results for signal-to-noise ratio and bit-rate gains in relation to the different profiles. For Corto both profiles produce the same geometric rate-distortion curves, while for Draco separate comparisons are offered for each speed preset. Similar to [31], when calculating the PSNR we use closest point correspondences and the bounding box's diagonal as the peak value[3]. While it is evident that coding speed comes at the expense of efficiency, we also observe smaller differences between Draco's slow and fast speed profiles.

4.4.2 Case B: Connectivity, Vertex Positions & Vertex Normals

In Fig. 7 relative codec performances for the case of compressing vertex normals along with mesh connectivity and geometry are given. The geometric distortions in this case for all codec profiles are the same as presented in subsection 4.4.1. While we have set a target of 1° (degree) RMS, *Corto1* could not achieve a better distortion performance than $\approx 6^\circ$ RMS. This behaviour has been mentioned in the discussion of Section 4.3.2. For *Corto1* we've set the quantization parameter to the value that led to the best distortion level it can achieve with the minimum possible bit-rate, since

3. Given that our test models are reconstructed in life-size scale after a uniform voxelization process, we expect consistent results across models.

TABLE 1
BD rates for Draco's and Corto's profiles. BD-SNR presents the quality gain for equal bit-rates, while BD-BR presents the percentage gain in bit-rate for the same quality.

	BD-SNR (dB)				BD-BR (%)			
	Corto12	***Draco2***	***Draco5***	***Draco9***	***Corto12***	***Draco2***	***Draco5***	***Draco9***
Corto12	N/A	-26.57 (1.82)	-23.1 (1.75)	-22.48 (1.97)	N/A	88.59 (6.99)	67.44 (5.66)	60.36 (5.51)
Draco2	26.57 (1.82)	N/A	4.12 (0.25)	5.6 (0.42)	-46.91 (2.01)	N/A	-10.39 (0.6)	-12.98 (0.86)
Draco5	23.1 (1.75)	-4.12 (0.25)	N/A	1.45 (0.41)	-40.22 (2.078)	11.6 (0.75)	N/A	-2.89 (0.97)
Draco9	22.48 (1.97)	-5.6 (0.42)	-1.45 (0.41)	N/A	-37.57 (2.19)	14.93 (1.13)	2.98 (1.03)	N/A

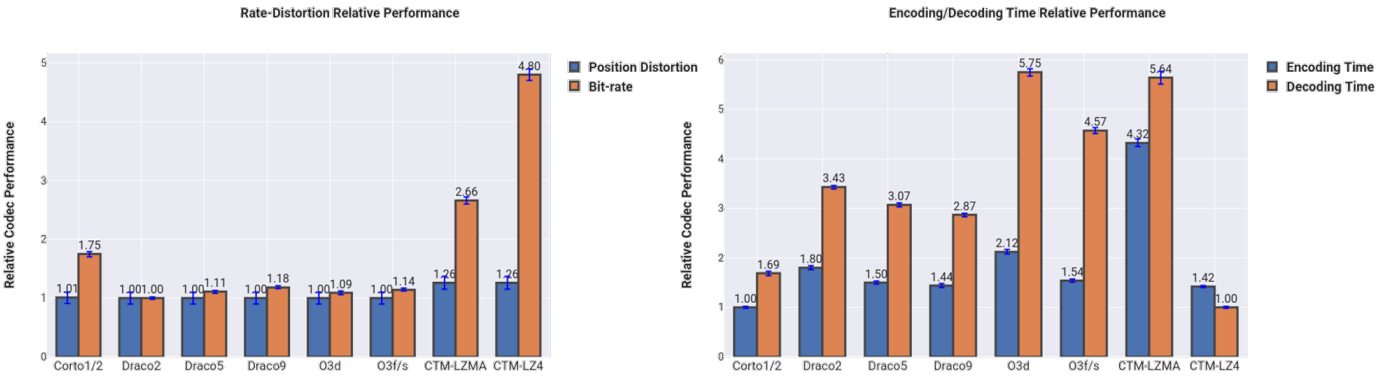

Fig. 6. Relative Comparison of Codec Performances: Geometric Distortion, Bitrate, Encoding and Decoding times when compressing connectivity along with geometry alone. The presented results correspond to a ≈ 0.00065 geometric Hausdorff distortion (with respect to the bounding box's diagonal).

further increasing the bit-rate did not significantly reduce the normal distortion. In addition, the normal distortion measurements taken for the previously mentioned profile have large standard deviation across the different models compared to all other codecs whose distortion performance is consistent for all models in the dataset.

Although *Draco2* produces the best bit-rate, it has a significant shortcoming in decoding time which is significantly worse than the rest of the codecs. However, *Draco5* is better than *O3d* in all aspects, while the same holds for *Draco9* over *O3f/s*. The fastest encoder and decoder is *Corto2*. Moreover, *Corto2* is consistently better than *Corto1* and both OpenCTM's profiles in all aspects. Finally, *Corto2* produces $\approx 40\%$ larger bit-rate compared to *Draco9* but is about $\approx 31.5\%$ and $\approx 50\%$ faster in encoding and decoding time, respectively.

4.4.3 Case C: Connectivity, Vertex Positions & Vertex Attributes

We present relative codec performances for the case of compressing mesh connectivity along with vertex geometry and attributes in Fig. 8. Geometric distortion across codecs is the same as in subsection 4.4.1. Corto produces about $\approx 50\%$ more RMS attribute distortion than the best performing codec (OpenCTM), while the others (Draco and O3dgc) produced $\approx 7\%$ more distortion than OpenCTM.

In this case, regarding O3dgc, the *O3f* performs closely to *O3d* in bit-rate terms, while for the encoding and decoding times it performs significantly better. *O3s* produces the lowest bit-rate (closely to *Draco9*) at the cost of increased time complexity.

Draco's profiles produce the best bit-rates. However, relative to one another, their encoding and decoding times vary disproportionately to the bit-rate gain. Draco's faster profile (*Draco9*) surpasses *O3d* and *O3f* in all aspects. Furthermore, it competes with *O3s* in bit-rate but outperforms it in encoding and decoding runtimes.

Corto, still, has the fastest encoder and the second fastest, but competitive, decoder after *CTM-LZ4*. Moreover, Corto performs better than *CTM-LZMA* in all aspects and better than *O3f* in all other aspects apart from decoding time. Comparing Corto with the fastest profile of Draco we see that Draco's profile produces about $\approx 33.3\%$ reduced bit-rate while it is $\approx 51\%$ and $\approx 65.8\%$ slower in encoding and decoding times, respectively. Thus, Corto offers a good balance between produced bit-rate and encoding/decoding runtimes.

4.4.4 Case D: Connectivity, Vertex Positions, Vertex Normals & Vertex Attributes

Finally, in Fig. 9 relative codec performances for the case of compressing connectivity along with all of vertex positions, normals and attributes are given. The distortion levels for this case are the same as the ones discussed in all the previous paragraphs of this section.

Regarding O3dgc, *O3f* produces slightly larger bit-rate that the other profiles of the same codec, but it has a significant improvement on encoding and decoding times. For Draco, once again we observe that the various profiles perform relatively close to one another in bit-rate, but they behave very much differently in encoding/decoding times with the profile performing better in bit-rate being considerably slower. When comparing Draco with O3dgc, we ob-

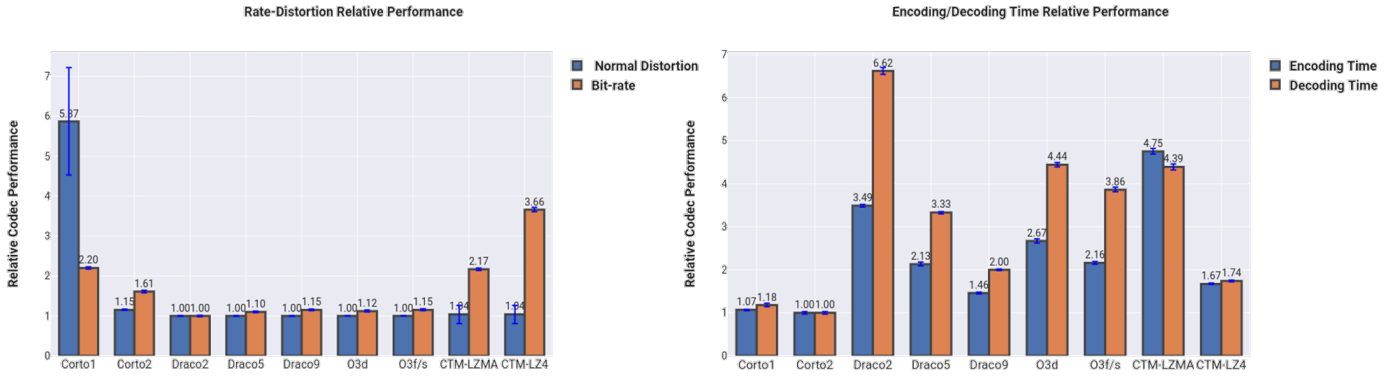

Fig. 7. Relative Comparison of Codec Performances: Normal Distortion, Bit-rate, Encoding and Decoding times when compressing connectivity along with geometry and normals. The presented results correspond to ≈ 0.00065 geometric Hausdorff distortion (with respect to the bounding box's diagonal) and $\approx 1^o$ normal angle distortion.

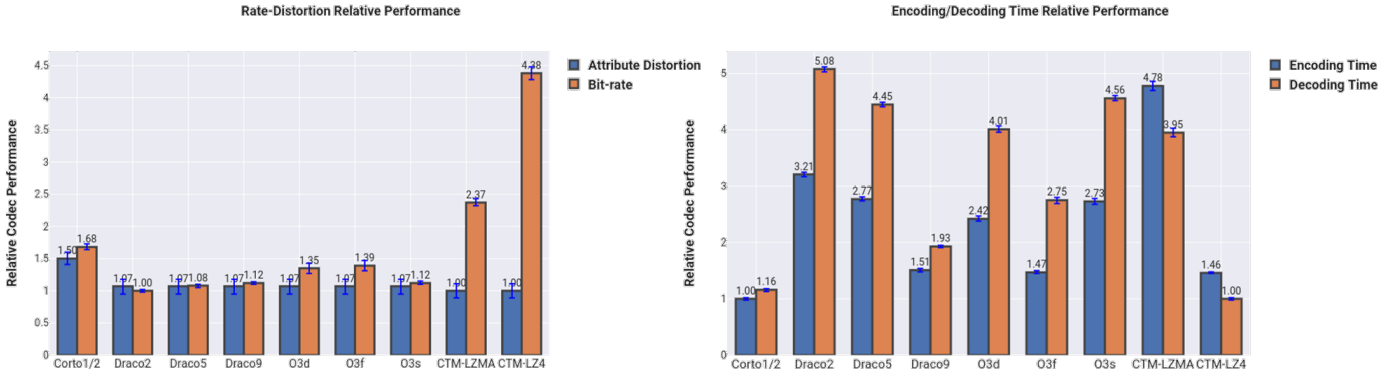

Fig. 8. Relative Comparison of Codec Performances: Attribute Distortion, Bit-rate, Encoding and Decoding times when compressing connectivity along with geometry and attributes. The presented results correspond to ≈ 0.00065 geometric Hausdorff distortion (with respect to the bounding box's diagonal) and ≈ 0.0145 attribute RMS distortion.

serve that the fastest profile of Draco (*Draco9*) outperforms all O3dgc's profiles in all aspects.

In this experiment, Corto has the fastest encoder and decoder. Further, it surpasses both OpenCTM's profiles in all aspects. Compared to *Draco9* it produces $\approx 43\%$ increased bit-rate but it is $\approx 33.7\%$ faster in encoding and $\approx 48.7\%$ faster in decoding.

4.4.5 Discussion on codec relative performances

Overall, in all the cases above, in terms of bit-rate, the best performing codecs are Draco and O3dgc with their various profiles performing close to one another. Corto's profiles are the best next followed by *CTM-LZMA* and *CTM-LZ4*. Regarding the encoding time, the best codec in all experiments is Corto, while for decoding time Corto remains the fastest decoder except from the case of decoding a mesh that contains only geometry and connectivity, in which case *CTM-LZ4* performs better. Evidently, we observe that when attributes (normals, texture indices, blending weights) are progressively added into the payload, *Corto2* performs slightly better in terms of bit-rate wrt the Draco baseline profile (*Draco2*), and that its runtime performance gain increases multifold. Moreover, O3dgc's slightly inferior performance in attribute coding is also observed when compared to Draco's baseline. Interestingly, OpenCTM's attribute coding appears to be the most performant in terms of relative gains, albeit still being overally inefficient.

5 Live Streaming Study

In this section of the paper we conduct a theoretical study on the performance of all codec profiles in a TI live streaming scenario by plugging the actual measurements we have acquired via benchmarking to a theoretical model of the typical TI pipeline. The theoretical model aims to determine the end-to-end latency of the streamed frames and the actual frame-rate at the receiver's side. The results of this theoretical analysis are influenced by the codecs' performances in all the aspects we have benchmarked previously (namely bit-rate, encoding and decoding runtimes). Futhermore, the analysis would not be complete if we didn't take into account the different possible network conditions that may apply in a TI scenario. Since none of the codecs participating in this study can decompress incomplete or erroneous compressed payloads, in the following study we assume that the employed network layer uses the Transmission Control Protocol (TCP) which guarantees both payload integrity and payload delivery.

In subsection 5.1 we introduce the TI pipeline along with the theoretical model that we use to opine about frame latency and frame-rate at the side of the receiver. In subsection 5.2 we present the calculation of the theoretical model's variables, while in subsection 5.3 we depict the results of the theoretical analysis for each codec profile and for various network conditions.

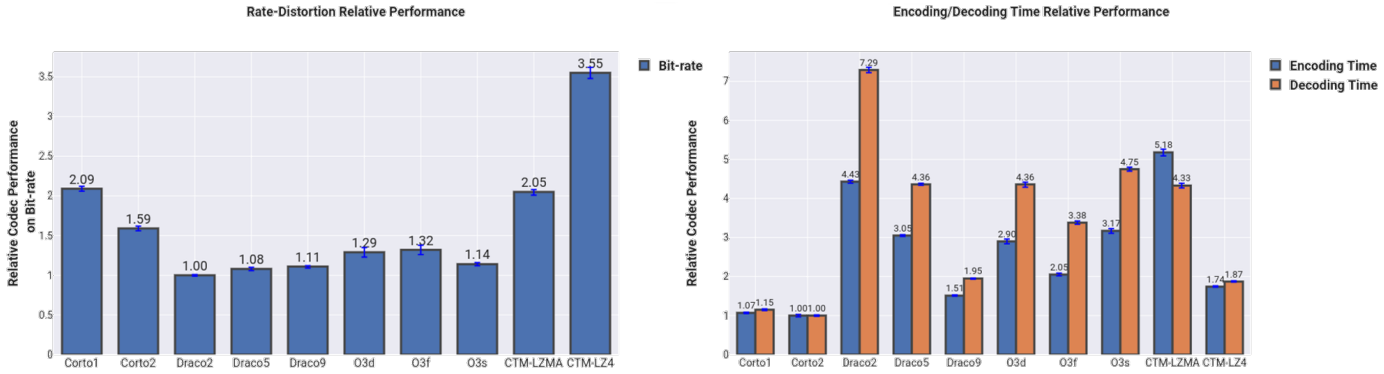

Fig. 9. Relative Comparison of Codec Performances: Bit-rate, Encoding and Decoding times when compressing connectivity along with geometry, vertex normals and vertex attributes. The presented results correspond to ≈ 0.00065 geometric Hausdorff distortion (with respect to the bounding box's diagonal), $\approx 1^o$ normal angle distortion and ≈ 0.0145 attribute RMS distortion.

5.1 The Tele-Immersion Pipeline and the Theoretical Model

In Fig. 10, the main components of a TI pipeline are given. The source is supposed to be a 3D reconstruction component that produces 3D meshes to be transmitted in real-time to remote parties. We do not impose any restrictions on the source (such as a specific output frame rate) because we want to study the capacity of the pipeline independent from source's characteristics. The 3D meshes produced by the source are subsequently compressed using anyone of the previously introduced codec profiles and then transmitted to a remote party which receives and decompresses the received payload prior to rendering. At the sender's side, the compression and transmission of the frames are assumed to be run in parallel (e.g. in separate threads with one thread compressing the next frame while the previous frame is transmitted to the network). When the compression of the current frame has been finished, the compressed representation is enqueued to the transmission queue ("Queue #1") in order to be sent to the network when the transmission of the previous frame finishes. Similarly, at the receiving side, the data received by the network are put in the decompression queue ("Queue #2") in order to be decompressed after the previously received frame has finished decompression. For the system to be as real-time as possible (i.e. minimize latency), both previously mentioned queues are assumed to store only one frame. If a new frame is enqueued while the previous frame has not been processed yet, the previous frame is dropped and its place is taken by the newer frame.

At each point of the pipeline we define auxiliary variable names that will help us with analyzing its end-to-end performance. These variables are:

- **cmprs_out**: The rate at which the encoder can provide compressed frames in frames per second.
- **cmprs_lat**: The time needed by the encoder to compress a single frame in seconds.
- **q1_wait_t**: The maximum time a frame will wait in the transmission queue ("Queue # 1") in seconds.
- **trans_push**: The maximum rate at which the compressed frames can be pushed into the network in frames per second
- **trans_lat**: The time, in seconds, that is required for a compressed frame to be transmitted over to the remote party.
- **trans_out**: The rate at which the frames arrive at the receiver from the network, in frames per second.
- **q2_wait_t**: The maximum time a frame will wait in the decompression queue ("Queue # 2") in seconds.
- **decmprs_push**: The maximum rate at which the received frames can be sent for decoding, in frames per second.
- **decmprs_lat**: The time, in seconds, needed by the decoder to decompress a single frame.
- **decmprs_out**: The output frame rate of the decoder in frames per second.

Following an analytic path to estimate the actual frame-rate and end-to-end latency of the TI pipeline can get cumbersome and complex. However, by making a few, simple, reasonable assumptions, we can calculate analytically lower and upper bounds of the pipeline's end-to-end latency and an estimate of its frame-rate.

We will first assume that the source is producing 3D meshes of approximate equal vertex and triangle count in each frame, which is a reasonable assumption when 3D reconstructing real humans standing at natural poses with a fixed resolution volumetric 3D reconstruction method.

Then, from the benchmark results that we have conducted for the various mesh codecs, we can acquire a good estimate on the encoding/decoding time needed by each codec in order to compress or decompress a frame. This is accomplished by first calculating an intermediate variable measured in "seconds per vertex" for the encoding and decoding operations, respectively. This variable is calculated by dividing the encoding / decoding time required to compress a mesh by the total number of vertices in the mesh and average across all models of the dataset, for given quantization parameters. With this data we can then estimate the average encoding and decoding time required by the respective codec in order to compress / decompress the meshes of approximately equal vertex count produced by the source.

In a typical TI pipeline the 3D reconstructed human meshes are further textured using the color images that were captured during frame acquisition. In our study, we account for an additional payload size that corresponds to 4 texture images (i.e. when using a 360° capturing setup of

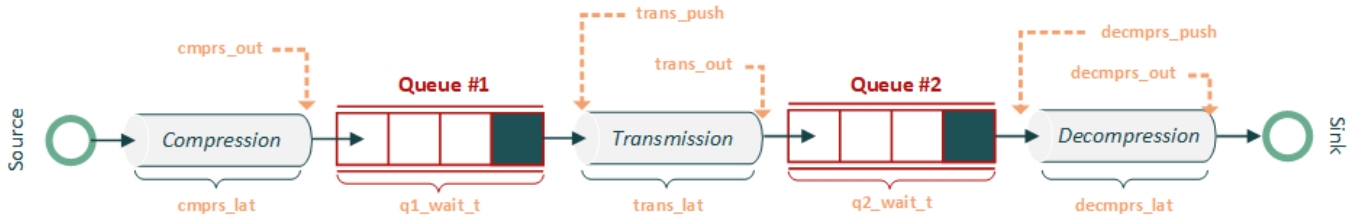

Fig. 10. Tele-Immersion Pipeline with variable names for live streaming study. Components of the Tele-Immersion pipeline are in **teal** (source and sink) or **gray** (processing stages), while variable names are depicted in **orange**.

four cameras) compressed separately, via the standard H.264 video codec [32] at a moderate quality. Concerning the additional processing time for compressing / decompressing the four textures, we will use typical encoding / decoding times that we calculated with internal benchmarking using the nVIDIA's GPU accelerated video codec NVENC [33] running on a GTX 1070 GPU.

For the network conditions we will assume a moderate, constant packet loss probability. Furthermore, since we assume TCP as the underlying network protocol, we will use the Mathis equation [34] to calculate the effective bandwidth of the network link based on the line's round trip time (RTT), bandwidth and packet loss probability. The Mathis equation is applicable to limit the effective bandwidth of the line when the maximum TCP congestion window (for the given packet loss) is smaller than the line's bandwidth delay product. Otherwise, the effective bandwidth of the link is equal to the line's bandwidth.

Finally, we will assume that the mesh compression / decompression components will be run on modern hardware, similar to what we used for these experiments (i.e. an Intel Core i7-7700K CPU clocked at 4.5 GHz - 4 physical cores, 8 logical threads - equipped with 32 GB of DDR4 RAM).

5.2 Calculating the variables of the TI pipeline's theoretical model

In order to calculate the aforementioned variables of the theoretical model of the TI pipeline, we will first need to define some additional supplementary variables that depend on the mesh codec used and the network line's characteristics. Those supplementary variables are given next:

- **cmprs_t**: The time needed to compress the 3D mesh and its four accompanying textures, in seconds. This time is the time needed by the mesh codec to compress the 3D mesh and the video codec to compress the four textures.
- **decmprs_t**: The time needed to decompress the 3D mesh and its four accompanying textures, in seconds.
- **EBW**: The effective bandwidth of the network line in Mbps.
- **frame_size**: The average compressed frame size in bytes with its payload corresponding to the sum of the compressed 3D mesh and the size of the four compressed accompanying textures.
- **RTT**: The line's round-trip time is seconds.

With the above definitions given, we are now able to calculate the variables of the TI pipeline's theoretical model:

$$\begin{aligned}
\mathbf{cmprs_rate} &= 1/\mathbf{cmprs_t}, \mathbf{cmprs_lat} = \mathbf{cmprs_t} \\
\mathbf{trans_push} &= 10^6 \times \mathbf{EBW}/8 \times \mathbf{frame_size} \\
\mathbf{q1_wait_t} &= \min\left(1/\mathbf{cmprs_rate}, 1/\mathbf{trans_push}\right) \\
\mathbf{trans_lat} &= 1/\mathbf{trans_push} + 0.5 \times \mathbf{RTT} \\
\mathbf{trans_out} &= \min\left(\mathbf{cmprs_rate}, \mathbf{trans_push}\right) \\
\mathbf{decmprs_push} &= 1/\mathbf{decmprs_t} \\
\mathbf{q2_wait_t} &= \min\left(1/\mathbf{decmprs_push}, 1/\mathbf{trans_out}\right) \\
\mathbf{decmprs_lat} &= \mathbf{dcmprs_t} \\
\mathbf{decmprs_out} &= \min\left(\mathbf{decmprs_push}, \mathbf{trans_out}\right)
\end{aligned}$$

According to the previous analysis, the approximate end-to-end frame-rate of the TI pipeline is equal to $\mathbf{decmprs\_out}$. A lower bound on the end-to-end latency is the quantity: $\mathbf{cmprs\_lat} + \mathbf{trans\_lat} + \mathbf{decmprs\_lat}$, while a respective upper bound is equal to $\mathbf{cmprs\_lat} + \mathbf{q1\_wait\_t} + \mathbf{trans\_lat} + \mathbf{q2\_wait\_t} + \mathbf{decmprs\_lat}$.

5.3 Codec Live-Streaming Performance

In this paragraph we apply the theoretical model we described in the previous subsection in order to obtain theoretical lower and upper bounds on the end-to-end latency of the TI pipeline and an estimate of the final achieved frame-rate. To accomplish this, we first set the scope of the evaluation. We are going to provide a study for the case of compressing and transmitting a full TVM representation with connectivity, geometry, normals and attributes as well as four texture images.

The meshes of our dataset that were generated by the 3D reconstruction method of [22] had an average vertex count of 16868, and thus we pick this number as our average mesh size. Subsequently, we set a target level of distortion for each generic vertex attribute that all codecs must meet for a fair comparison. The distortion levels that we have set have already been discussed in the previous section of the paper. More specifically, for geometry we've set a target distortion level of ≈ 0.00065 Hausdorff distance with respect to the bounding box diagonal. For normals, we set the RMS distortion to 1° and for attributes to 0.0145 RMS.

From the benchmark analysis that we have performed and discussed in the previous section, we are able to do inverse calculation and compute the quantization parameter values that lead to this average distortion behavior for each codec. For the obtained quantization parameter values we can calculate the estimated resulting bit-rate and encoding

/ decoding time performance of each codec. Thus, the **frame_size** variable can be estimated by the calculated bit-rate, while the encoding and decoding times of the codec are calculated according to the methodology described in Section 5.1.

In addition, we assume that the four accompanying textures of each mesh are half the Full HD resolution (960×540 pixels) and are compressed independently using the H.264 video codec, configured in its low-latency / high performance "zerolatency" mode (i.e. not using B frames) and constant quality across frames with quantization parameter 31, which is equivalent to per-frame JPEG compression with 50% quality in PSNR sense (determined after experimentation with textures from our dataset). An extensive benchmarking experiment that we have conducted in our labs concerning the texture compression has shown that for this texture size and compression parameters the average encoding time for four textures is 4 ms (1ms per texture) while for decoding is 2 ms producing an average size per texture (per frame) of 1KB (total of 4KB for four textures). The implementation of the H.264 encoder that we used was hardware accelerated in GPU [33].

Regarding the network conditions, we pick a fixed, moderate, packet loss probability of $p = 0.0005$ for all of our experiments. For the network line's bandwidth we evaluate on a typical value of 100Mbps . Moreover, we consider RTT of 1ms, 20ms and 50ms which implies that the remote party may be in the same city, country or continent as the sender, respectively. The actual numbers for the upper and lower bounds of the end-to-end latency as well as the expected frame-rate (in frames per second - fps) for each codec profile are given in Table 2.

Discussion on codec live streaming performance

Regarding the RTT of 1ms, we observe that among the same codec, the fastest profiles perform better in both latency bound terms and expected frame rate. In particular, *Draco9*, *O3f* and *CTM-LZ4* perform better than the other profiles of the respective codecs. Corto's profiles perform similarly. Overall, in terms of end-to-end latency and frame-rate the best performance is achieved by *Corto2*.

For the case of 20 ms RTT , the fastest profiles of each codec performs better than other profiles of the same codec in all aspects. However, in contrast to the previous case, the codec with the lowest lower bound end-to-end latency and highest frame rate is *Draco9*, while the lowest upper bound end-to-end latency is achieved by *Corto2*, but not significantly less than *Draco9*.

Finally, when considering a network line with RTT of 50 ms, for each codec, the profile that achieves the highest compression leads to the highest frame-rate. On the other hand, the fastest profiles generally achieve better performance in terms of end-to-end latency. Overall, the best performing codec is Draco with "speed - 2" achieving the highest frame-rate and "speed - 9" achieving the lowest end-to-end latency.

To sum up, a general observation is that the processing speed of the mesh codecs is more important when the network line's RTT is small, while for larger values of RTT the best performing codecs are the ones having a good trade-off between processing time and compression rate. This can easily be explained by the theoretical model we have discussed in Subsection 5.1. When effective bandwidth is high, the frame rate is only affected by the compression time (since compression time is always worse than decompression time for all codec profiles). Furthermore, in the same case, the transmission latency is negligibly affected by the frame size, adding to the fact that compression ratio here is less important. On the other hand, when the effective bandwidth is low, the frame rate is inversely proportional to the frame size, and latency is dominated by transmission time, which is proportional to the frame's size. Thus, in that case, the RD performance matters more than runtime performance.

6 Conclusion

In this paper we have conduced an extensive, systematic and consistent benchmarking in terms of bit-rate, distortion and processing time of four Open-Source static 3D mesh codecs namely Corto, Draco, O3dgc and OpenCTM. In contrast to other works, our work examines thoroughly the performance of the codecs not only in compressing geometry along with connectivity, but also in compressing vertex normals and attributes.

Firstly, we evaluated the codecs in rate-distortion terms, accounting for geometry, normal as well as attribute distortion. Subsequently, we set target levels of distortion in all of the aforementioned aspects, that led to a compressed representation of good quality (at least in objective terms) but still significantly compressible. For these preset distortion levels, we evaluated the performance of the codecs in relation to one another, in the aspects of bit-rate and processing time. We've concluded that by the relative analysis it is not easy to opine on which codec performs best, since none of the codecs tops in all of the bit-rate, encoding and decoding time aspects. Thus, we continued our investigation on the performance of the codecs by examining their theoretical performance in the case they were employed in a tele-immersive interactive live streaming scenario. We created a theoretical model of a TI pipeline and analytically computed the end-to-end latency lower and upper bounds as well, as the expected frame-rate for some common network conditions when utilizing each one of those codecs. The values that we fit to the TI pipeline's theoretical model were obtained by previously conduced extensive benchmarking.

Overall, we found that in the live streaming scenario, for each profile of O3dgc there does exist one profile of Draco that performs better in all of the relevant codec aspects. Furthermore, except of the case of compressing just geometry along with connectivity, the Corto codec always performes better than the OpenCTM's profiles in all terms. Our live streaming analysis showed that choosing between Corto and Draco in a TI pipeline should be a decision based on network conditions, with Corto performing best on network setups with low RTT, while Draco being better when the line's RTT increases. The results of this study may be used to help designers of Tele-Immersive systems to chose the best 3D mesh codec for their application. In addition, the aforementioned analysis can be used as a baseline for future 3D compression research, as it is evident that there is still room for improvement, given that according to our theoretical analysis, when the networking parameters

TABLE 2
Expected frame-rate and end-to-end latency lower (LB) and upper (UB) bounds for each codec profile in a live streaming scenario for a set of networking conditions with line bandwidth 100 MBps and RTT of 1ms, 20ms and 50ms. Each case's effective bandwidth is also reported.

	Line BW: 100Mbps, RTT: 1ms, EBW: 100Mbps			Line BW: 100Mbps, RTT: 20ms, EBW: 26.12Mbps			Line BW: 100Mbps, RTT: 50ms, EBW: 10.47Mbps		
Codec Profile	**Latency LB**	**Latency UB**	**Frame-Rate**	**Latency LB**	**Latency UB**	**Frame-Rate**	**Latency LB**	**Latency UB**	**Frame-Rate**
Corto1	25.65	38.75	82.95	57.49	74.74	33.07	117.83	135.09	13.23
Corto2	**24.58**	**37.02**	**85.82**	55.70	**72.15**	34.17	114.60	131.04	13.66
Draco2	65.69	92.99	26.39	89.14	130.40	26.39	132.47	192.73	**21.18**
Draco5	47.32	66.83	36.60	71.85	106.39	36.60	117.36	158.88	19.66
Draco9	28.94	41.81	64.19	**53.80**	76.82	**48.12**	**99.97**	**122.99**	19.24
O3d	47.12	67.58	38.23	74.40	112.65	38.23	125.51	165.85	16.61
O3f	38.05	55.91	50.78	65.68	96.82	40.73	117.51	148.65	16.29
O3s	49.62	70.49	35.38	74.91	111.56	35.38	121.96	165.50	18.72
CTM-LZMA	67.95	91.82	22.93	105.07	156.54	22.93	176.12	233.82	10.70
CTM-LZ4	41.75	65.68	57.73	98.47	123.02	15.64	209.33	233.89	6.25

used resemble the actual Internet more closely, sub-optimal frame-rates are being achieved by the majority of codecs.

ACKNOWLEDGMENTS

This work has been supported by the EU's H2020 programme 5G-MEDIA (GA 761699).

REFERENCES

[1] A. Maimone and H. Fuchs, "Encumbrance-free telepresence system with real-time 3d capture and display using commodity depth cameras," in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*. IEEE, 2011, pp. 137–146.

[2] S. Beck, A. Kunert, A. Kulik, and B. Froehlich, "Immersive group-to-group telepresence," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 4, pp. 616–625, 2013.

[3] N. Zioulis, D. Alexiadis, A. Doumanoglou, G. Louizis, K. Apostolakis, D. Zarpalas, and P. Daras, "3D tele-immersion platform for interactive immersive experiences between remote users," in *Image Processing (ICIP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 365–369.

[4] S. Orts-Escolano, C. Rhemann, S. Fanello, W. Chang, A. Kowdle, Y. Degtyarev, D. Kim, P. Davidson, S. Khamis, M. Dou, et al., "Holoportation: Virtual 3d teleportation in real-time," in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. ACM, 2016, pp. 741–754.

[5] A. Doumanoglou, D. S. Alexiadis, D. Zarpalas, and P. Daras, "Toward real-time and efficient compression of human time-varying meshes," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 12, pp. 2099–2116, Dec 2014.

[6] S. R. Han, T. Yamasaki, and K. Aizawa, "Time-varying mesh compression using an extended block matching algorithm," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 11, pp. 1506–1518, Nov 2007.

[7] T. Yamasaki and K. Aizawa, "Patch-based compression for time-varying meshes," in *2010 IEEE International Conference on Image Processing*, Sept 2010, pp. 3433–3436.

[8] "MPEG-I, 2018. Coded Representation of Immersive Media. ISO/IEC 23090," https://mpeg.chiariglione.org/standards/mpeg-i.

[9] "MPEG-PCC, 2018. Coded representation of Point Clouds," https://mpeg.chiariglione.org/standards/mpeg-i/point-cloud-compression.

[10] R. Mekuria, K. Blom, and P. Cesar, "Design, implementation, and evaluation of a point cloud codec for tele-immersive video," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 4, pp. 828–842, April 2017.

[11] D. Thanou, P. A. Chou, and P. Frossard, "Graph-based compression of dynamic 3d point cloud sequences," *IEEE Transactions on Image Processing*, vol. 25, no. 4, pp. 1765–1778, April 2016.

[12] "Google Draco," https://github.com/google/draco, accessed: 2018-06-07.

[13] "Corto," https://github.com/cnr-isti-vclab/corto, accessed: 2018-06-07.

[14] "Open 3D Graphics Compression (O3DGC)," https://github.com/amd/rest3d/tree/master/server/o3dgc, accessed: 2018-06-07.

[15] "OpenCTM," http://openctm.sourceforge.net/, accessed: 2018-06-07.

[16] J. Rossignac, "Edgebreaker: Connectivity compression for triangle meshes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 1, pp. 47–61, Jan. 1999.

[17] F. Ponchio and M. Dellepiane, "Fast decompression for web-based view-dependent 3d rendering," in *Proceedings of the 20th International Conference on 3D Web Technology*, New York, NY, USA, 2015, Web3D '15, pp. 199–207, ACM.

[18] K. Mamou, T. Zaharia, and F. Prêteux, "TFAN: A low complexity 3D mesh compression algorithm," *Comput. Animat. Virtual Worlds*, vol. 20, no. 2-3, pp. 343–354, June 2009.

[19] Faxin Yu, Hao Luo, Zheming Lu, and Pinghui Wang, "3d mesh compression," in *Three-Dimensional Model Analysis and Processing*, pp. 91–160. Springer, 2010.

[20] J. Peng, C-S. Kim, and C. Jay Kuo, "Technologies for 3d mesh compression: A survey," *J. Vis. Comun. Image Represent.*, vol. 16, no. 6, pp. 688–733, Dec. 2005.

[21] A. Maglo, G. Lavoué, F. Dupont, and C. Hudelot, "3d mesh compression: Survey, comparisons, and emerging trends," *ACM Comput. Surv.*, vol. 47, no. 3, pp. 44:1–44:41, Feb. 2015.

[22] D. Alexiadis, A. Chatzitofis, N. Zioulis, O. Zoidi, G. Louizis, D. Zarpalas, and P. Daras, "An integrated platform for live 3D human reconstruction and motion capturing," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 4, pp. 798–813, 2017.

[23] William E. Lorensen and Harvey E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. 1987, SIGGRAPH '87, pp. 163–169, ACM.

[24] Quirin Meyer, Jochen Süßmuth, Gerd Sußner, Marc Stamminger, and Günther Greiner, "On floating-point normal vectors," in *Computer Graphics Forum*. Wiley Online Library, 2010, number 4 in EGSR'10, pp. 1405–1409.

[25] C. Touma and C. Gotsman, "Triangle mesh compression," Dec. 26 2000, US Patent 6,167,159.

[26] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[27] Y. Collet et al., "Lz4: Extremely fast compression algorithm," *code. google. com*, 2013.

[28] P Cignoni, C. Rocchini, and R. Scopigno, "Metro: measuring error on simplified surfaces," 1998.

[29] Gisle Bjontegaard, "Calculation of average psnr differences between rd-curves," *VCEG-M33*, 2001.

[30] Gisle Bjontegaard, "Improvements of the bd-psnr model, vceg-ai11," in *ITU-T Q. 6/SG16, 34th VCEG Meeting, Berlin, Germany (July 2008)*, 2008.

[31] Philip A Chou, Eduardo Pavez, Ricardo L de Queiroz, and Antonio Ortega, "Dynamic polygon clouds: Representation and compression for vr/ar," *arXiv preprint arXiv:1610.00402*, 2016.

[32] Iain E. Richardson, *The H.264 Advanced Video Compression Standard*, Wiley Publishing, 2nd edition, 2010.

[33] "NVIDIA NVENC," https://developer.nvidia.com/nvidia-video-codec-sdk.

[34] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the tcp congestion avoidance algorithm," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 3, pp. 67–82, 1997.

Alexandros Doumanoglou received his diploma in Electrical & Computer Engineering from the Aristotle University of Thessaloniki (AUTh) in 2009. His thesis was about the study and construction (in hardware and software) of a passive acoustic radar. In the past, he has been involved in in-depth study of several fields of computer science, with software security, cryptography, algorithms and theory of computation, 3D graphics and artificial intelligence in computer games being the most notable. He is fond of applied informatics and engineering. He has been working in the Information Technologies Institute (ITI) as a research assistant since April 2012. His main research interests include computer vision, pattern recognition, 3D reconstruction, 3D graphics, mesh compression, signal processing and mathematical optimization.

Petros Drakoulis received his BSc in IT Engineering from Alexander TEI and his MSc in Digital Media from Aristotle University of Thessaloniki. In 2018, he joined information technologies institute (ITI) of the Greek national centre for research and technological advancement (CERTH) as a research assistant. His interests include software engineering, computer vision and machine learning.

Nikolaos Zioulis is an Electrical and Computer Engineer (Aristotle University of Thessaloniki, 2012) working in the Information Technologies Institute (ITI) of the Centre for Research and Technology Hellas (CERTH) since October 2013. His interests include 3D processing and graphics, particularly performance oriented real-time computer vision. Having been involved in various research projects, his research interests lie in the intersection of computer vision and graphics technologies and, more specifically, 3D capturing and rendering, 3D scene understanding and tele-immersive applications.

Dimitrios Zarpalas has joined the Information Technologies Institute in 2007, and is currently working as a post-doctoral Research Associate. His current research interests include real time tele-immersion applications (3D reconstruction of moving humans and their compression), 3D computer vision, 3D medical image processing, shape analysis of anatomical structures, 3D object recognition, motion capturing and evaluation, while in the past has also worked in indexing, search and retrieval and classification of 3D objects and 3D model watermarking. His involvement with those research areas has led to the co-authoring of 3 book chapter, 1 articles in refereed journals and 42 papers in international conferences. He has been involved in more than 10 research projects funded by EC, and Greek Secretariat of Research and Technology. He is a member of the Technical Chamber of Greece.

Petros Daras is a Principal Researcher Grade A', at the Information Technologies Institute of the Centre for Research and Technology Hellas. He received the Diploma in Electrical and Computer Engineering, the MSc degree in Medical Informatics and the Ph.D. degree in Electrical and Computer Engineering all from the Aristotle University of Thessaloniki, Greece in 1999, 2002 and 2005, respectively. He is the head researcher of the Visual Computing Lab coordinating the research efforts of more than 35 scientists. His research interests include 3D media processing and compression, multimedia indexing, classification and retrieval, annotation propagation and relevance feedback, bioinformatics and medical image processing. He has co-authored more than 160 papers in refereed journals and international conferences, and has been involved in more than 30 national and international research projects.