Compression of Time-Varying Textured Meshes using Patch Tiling and Image-based Tracking

Jean-Eudes Marvie, Maja Krivokuća, Céline Guede, Julien Ricard, Olivier Mocquard, François-Louis Tariolle
InterDigital INC, Rennes, France
jean-eudes.marvie@interdigital.com, maja.krivokuca@interdigital.com, celine.guede@interdigital.com,
julien.ricard@interdigital.com, olivier.mocquard@interdigital.com, francois-louis.tariolle@interdigital.com

***Abstract*—This paper presents our answer to the dynamic mesh compression Call for Proposals (CfP) that was recently launched by the MPEG 3D Graphics Coding (MPEG-3DGC) group. The proposed method begins with a per-frame mesh decimation based on quadric error metrics. The decimated mesh geometry and topology are then encoded per frame, by using a state-of-the art static mesh coder (such as Draco, for example). The associated texture map coordinates (*uv*) can either also be directly encoded using the same static mesh coder, or an optional mesh segmentation into sub-meshes ("patches") of approximately equal size is applied. For each patch, a new local *uv* parameterization is computed, meaning that the original *uv* texture coordinates do not need to be encoded, as the same parameterization is recomputed by the decoder. The patches are then organised into a regular grid in a global *uv* coordinate system, and the input mesh texture map is reprojected onto the corresponding patch tiles. Intra-frame reorganisation of the texture tiles can then be applied in the first frame of each Group of Pictures (GOP), followed by inter-frame reorganisation for the other frames in the same GOP based on tile tracking using image-based distance metrics. These reorganisations improve spatial and temporal correlations in the reprojected texture maps, allowing these texture maps to be more efficiently coded by standard 2D video codecs than the original texture maps. The proposed method shows notable improvements in rate-distortion performance over the anchor codec used in the CfP, both for the geometry and colour coding, and for the all-intra and random-access lossy coding test conditions.**

***Index Terms*—volumetric video, mesh compression, dynamic meshes, textured meshes, time-varying meshes**

I. Introduction

Volumetric video is one of the newest forms of multimedia that promises to take the viewer one step closer towards a more realistic and more immersive viewing experience than the traditional two-dimensional (2D) video. A volumetric video consists of a sequence of frames, where each frame is a static three-dimensional (3D) representation of a real-world object or scene captured at a different point in time. While a number of possible 3D representations could be used, nowadays the most common are 3D *point clouds* and 3D *mesh models*. Both of these representations consist of a set of point (*vertex*) positions in 3D space, (x, y, z), which are known as the model's *geometry*. Each vertex could optionally have additional *attributes* associated with it, most typically (R,G,B) colours. A 3D mesh additionally contains an explicit definition of the 3D object's surface, which comes in the form of *connectivity* data. The connectivity is a set of straight edges between the vertices, which form flat polygons (usually triangles) called *faces*. The collection of faces, together with the vertex positions, provides an approximation of the underlying 3D object's surface. The colours or other attributes of a 3D mesh could either be defined per point, or per face. Colours could also be associated with the mesh surface by reparameterizing the mesh onto 2D regions of a plane, then using the corresponding set of parametric coordinates (called *uv* coordinates) to map a 2D texture onto the mesh. The latter is commonly called a *textured mesh* and it is what we focus on in this paper.

Regardless of the underlying 3D representation used, volumetric videos are heavy with data and require efficient compression techniques in order to be usable in practice. Recently, the 3D Graphics Coding subgroup of the Moving Picture Experts Group (i.e., MPEG-3DGC) launched a Call for Proposals (CfP) [1], to address the problem of compressing dynamic (*time-varying*) textured meshes (TVMs) in the context of volumetric video. The aim of this paper is to describe our proposed solution to this CfP. While earlier MPEG standards do exist for coding dynamic 3D meshes [2], these standards cover only so-called *animated meshes*, which are sequences of mesh models (usually computer-generated) where the number of vertices, the mesh connectivity, and the associated uv coordinates remain constant, and only the vertex positions change over time. If the animated meshes contain texture information, the texture map remains constant over time (i.e., there is only one texture frame for the entire mesh sequence). The dynamic meshes that we are interested in compressing now can have a variable number of vertices, and therefore a different connectivity and topology, and different uv texture coordinates, for each frame. The texture map content for these meshes is also usually updated at each frame, such that the colour attributes are a *sequence* of texture maps. Such meshes are more representative of real-world captures, where the 3D model(s) in each frame are generated independently; however, for this reason, TVMs are also much heavier in terms of data content than animated meshes. The coding of TVMs is a challenging problem that started to be addressed in the research community almost two decades ago [3], [4] (usually for dense meshes with colours per vertex, or no colours at all, but not for textured meshes), but is still very much in its infancy. The most difficult aspect of compressing such data is the lack of explicit correspondences in the connectivity and geometry of the mesh models across different frames. Tracking algorithms that attempt to find such correspondences are often

costly and time-consuming, and therefore not practical to use for a standalone TVM codec. These instabilities also lead to unstable texture map layouts, which are difficult to compress efficiently using a video coder. All of these challenges, plus the fact that the texture map for TVMs changes at each frame, unlike the one constant texture map for animated meshes, means that the results for existing TVM codecs are still far from the compression rates achievable for constant-connectivity animated meshes.

In this paper, we propose a new TVM coding solution that achieves substantially better *rate-distortion* (R-D) performance results than the anchor codec used for the MPEG-3DGC mesh coding CfP [1]. Our solution is based on a per-frame mesh decimation using the algorithm from [5], followed by a per-frame encoding of the decimated mesh geometry and topology by using a state-of-the-art mesh codec such as the Draco [6] implementation of Edgebreaker [7]. The associated texture *uv* coordinates can either also be encoded by Draco, or an optional mesh segmentation into sub-meshes ("patches") can be applied. In the latter case, these patches' texture uv coordinates are reparameterized such that they do not need to be encoded, as the same reparameterization is computed at the decoder. A texture map reprojection onto the patch tiles in the new uv coordinate system, followed by intra- and inter-reorganisation of these texture tiles using image-based distance metrics, allows an improved spatial and temporal correlation that enables the reprojected texture maps to be coded more efficiently by standard 2D video codecs than the original texture maps.

The rest of this paper is organised as follows. Section II covers the prior work in TVM compression, Section III introduces our proposed solution, Section IV presents our key experimental results, and Section V concludes the paper.

II. Related Work

In some of the earliest work on time-varying mesh compression [3], [8], researchers proposed the use of surface flattening (or unfolding) techniques for 3D meshes. These methods cut open a 3D mesh, then project it onto 2D images, which can be encoded with conventional 2D video coders such as H.264/AVC [9]. Several other TVM compression methods [4], [10], [11] are based on the subdivision of 3D meshes into blocks, to extend the idea of 2D block matching from conventional 2D video coding to 3D. In [12], [13], sub-meshes (patches) are used instead of 3D blocks. However, none of these methods have been demonstrated to work for textured meshes, but rather for meshes with colours per vertex, or colourless meshes.

More recently, TVM coding solutions have begun to appear [14], [15] that are based on the MPEG Video-based Point Cloud Coding (V-PCC) standard from the V3C framework [16]. However, [14] does not work for textured meshes either (only for dense meshes with per-vertex colours), and has also not been demonstrated for *sequences* of time-varying meshes; only for a single mesh. The authors also showed that their method is usually outperformed by Draco [6]. The method in [15] works on both, meshes with per-vertex colours and textured meshes. In [15], each vertex position of a 3D mesh is projected onto a pixel position in a 2D patch image, similarly to how 3D point clouds are coded with V-PCC, but additionally the surface of the 3D mesh is also projected onto the 2D patch projection plane by using rasterization. This produces a dense image representing the mesh connectivity, which is suitable for video coding. Occupancy, geometry, and attributes are encoded as usual by V-PCC. However, similarly to [14], the method in [15] has also been shown to be outperformed by Draco for geometry compression, while for colour compression the performance is comparable (or in some cases a little better) to the case when the original texture map is encoded using HEVC [17] on top of a reconstructed geometry using Draco.

It is therefore clear that, in the limited work that currently exists on TVM compression, there is still no suitable solution for the efficient compression of time-varying textured meshes. The key component of the solution that we propose in this paper, which is different to existing TVM compression algorithms, is a method to stabilise the input texture atlases spatially within a frame and also temporally across frames, and thereby allow the resulting texture maps to be coded efficiently with existing 2D video coders. The heart of our codec is illustrated in Fig. 1. More details on the proposed method are explained in the following section.

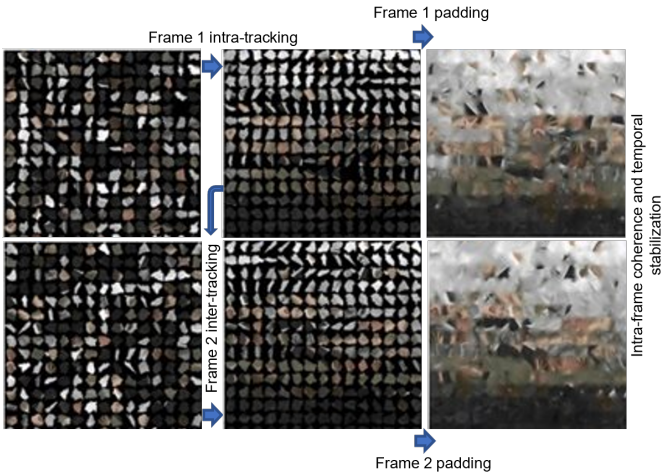

Fig. 1. Example of resulting texture frames after our processes from Sections III-A2 to III-A4 are applied on the "basketball" mesh sequence [1].

III. Proposed Codec for Time-Varying Meshes

Figs. 2 and 3 depict the proposed encoder and decoder architectures. The key processing blocks are described in the sub-sections below.

A. Encoder

1) Mesh simplification: The encoder begins with a mesh simplification (*decimation*), to reduce the number of triangles in the input mesh but without visibly degrading the mesh

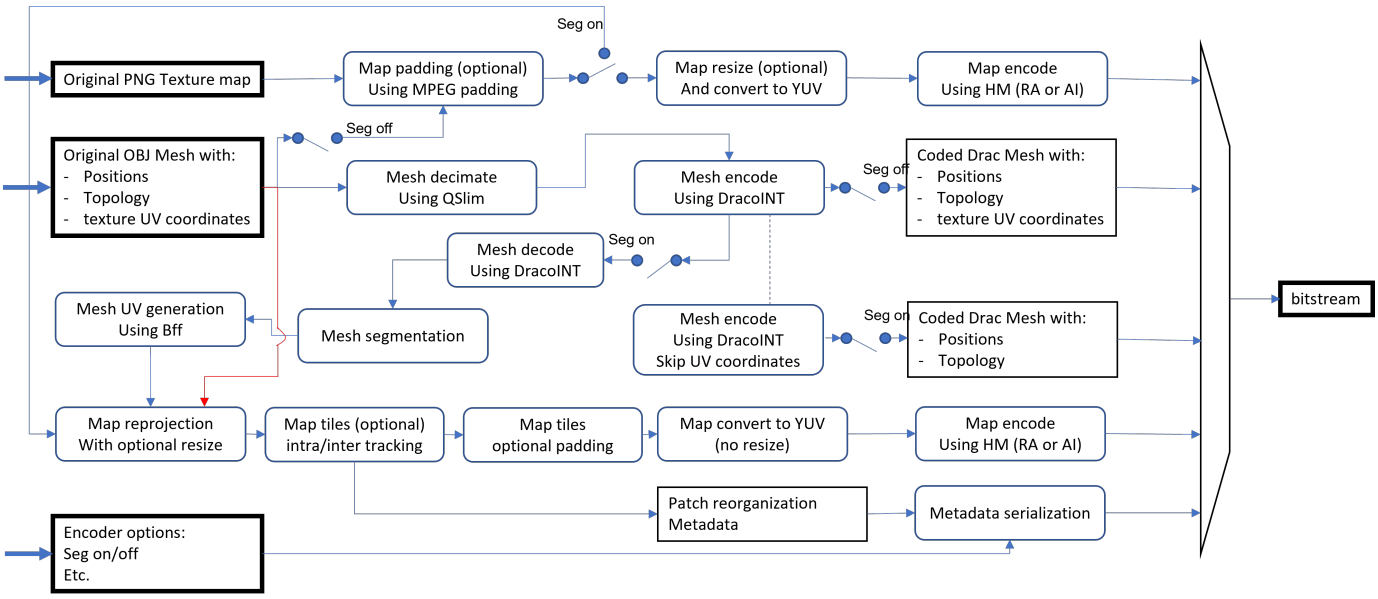

Fig. 2. Proposed encoder architecture.

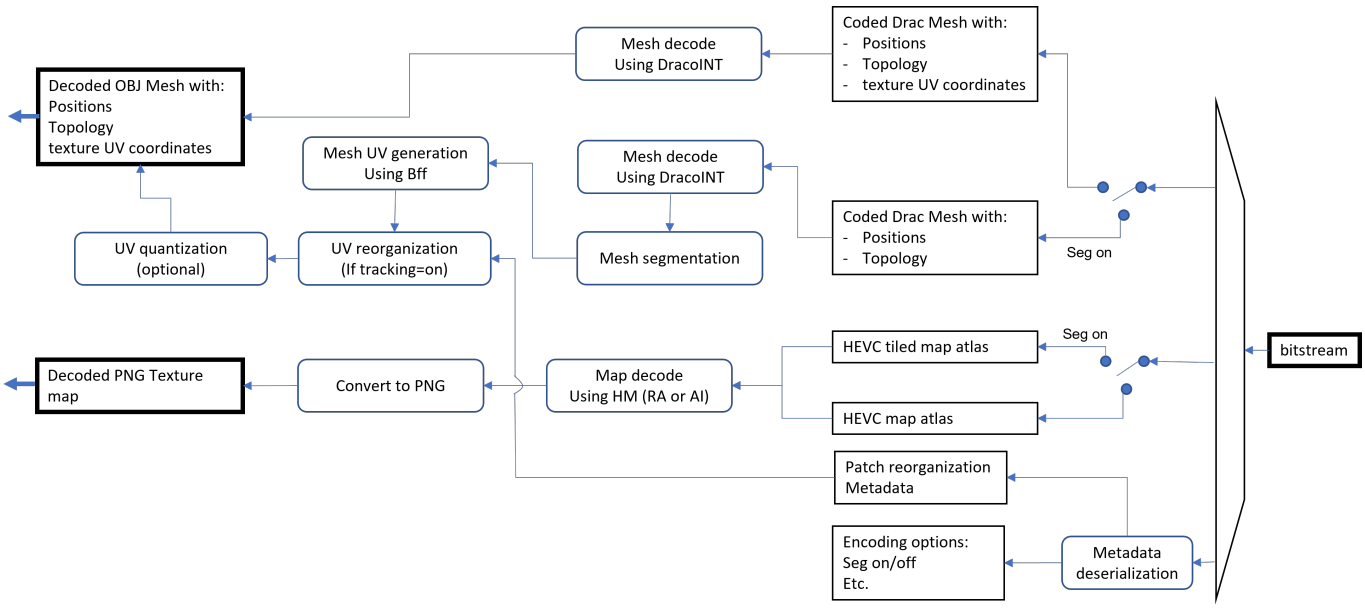

Fig. 3. Proposed decoder architecture.

quality. A number of possible decimation methods could be used, but our current implementation is based on the well-known algorithm from Garland and Heckbert [5]. The simplified mesh geometry, connectivity, and texture map *uv* coordinates can then all be encoded directly by a state-of-the art mesh encoder such as Draco [6], or only the geometry and connectivity are coded by Draco but the *uv* coordinates are not encoded at all. In the latter case, a segmentation into sub-meshes ("patches") of approximately equal size is the next step, explained below.

2) Mesh segmentation into patches: To obtain the mesh patches, we use a similar principle to [18], except that the "seeds" are chosen differently in our work. The seeds are representative points that are used to cluster mesh faces to their nearest seed, such that each seed becomes one sub-mesh (patch). In [18], each new seed is chosen so as to maximise the average distance to all the existing seeds in the same connected component. However, this does not work well for our proposed system, as it leads to non-uniform patches, but we wish to have approximately uniform patches in order to make patch reorganisation and tracking easier. Therefore, in our solution, each new seed is added to maximise the *minimal* distance (using Dijkstra's algorithm [19]) to all the existing seeds in the same connected component. The first seed is chosen by

taking the triangle whose centre of mass is nearest to the mesh's centre of mass. An optional refinement step is also possible, where we select, for each patch, a new seed amongst its faces, which is closest to the centre of mass of the patch. However, we found that even without this refinement, we can already obtain good results with a lower computational time complexity; therefore, the results in Section IV do not include this refinement step.

3) Patch uv reparameterization: Once the patches are generated, we compute for each patch a new local *uv* parameterization. The advantage of this step is that the original *uv* texture coordinates do not need to be encoded, as the same reparameterization can be computed locally at the decoder. In our implementation, the new *uv* coordinates for each patch are generated using the *Boundary First Flattening* (BFF) algorithm and software [20] (but this could also be replaced by other algorithms, e.g., [21], [22]). We obtain for each patch a *uv* parameterization with *uv* coordinates set between 0.0 and 1.0 in each axis direction (u and v). Next, all the reparameterized patches are organised into a regular grid in a *global uv* coordinate system, by using scaling and translation of the local patch *uv* coordinates. The input mesh texture map is then reprojected onto the corresponding patch tiles (e.g., see Fig. 1 (Left)). Two paths are possible for the texture reprojection: either reprojecting the textures by using the *uv* coordinates of the decimated (and decompressed) mesh, or performing a colour transfer (red line in Fig. 2) between the original input mesh and the decompressed (decimated) mesh. To organise the patches in the global coordinate system, we simply take the patches in the order that they are generated by the segmentation process. The order of the patches is not critical here, since the corresponding texture tiles will be reorganised later (see Section III-A4). Also note that the BFF *uv* generator can be executed with different modes (e.g., target patch shapes). Our method works with each mode, but since the "cone" mode generates the lowest distortions in the reprojections, we use this one for our CfP response.

4) Intra- and inter-frame texture tile reorganisation: Looking at the example in Fig. 1 (Left), we can see that the empty regions (black areas) between patches introduce strong signal discontinuities at the patch boundaries. Even when patches are not separated by empty regions, the transition from one patch to another may present strong colour or luminance differences. A common solution to reduce such sharp signal transitions is to apply some padding to fill the empty areas with data that suits the encoder, e.g., by smoothing out the transitions between patches. In our work, we use the "harmonic" padding from the MPEG V-PCC standard [16]. However, this padding alone is not sufficient. In Fig. 1, we can see that the texture tiles are also not necessarily arranged in a way where they have a (strong) coherence with their neighbouring tiles. Therefore, before applying the padding, we can first perform an intra-frame tile reorganisation to minimise the signal discontinuities between neighbouring tiles. This leads to padding gradients of smaller amplitude, thereby improving the compression of the texture map by the 2D video coder that relies on inter-block predictions. The intra-frame reorganisation of the texture tiles (corresponding to the mesh patches in the global *uv* coordinate system, described in Section III-A3) is applied in the first frame of each Group of Pictures (GOP). It is based on scanning the grid of tiles from the bottom left corner to the top right corner, and at each iteration searching amongst the remaining unorganised texture tiles to find the tile that would minimise the inter-tile transitions (in terms of *mean squared error* (MSE)) with the current tile and its immediate neighbours (see Fig. 4). The MSE between two tiles is measured between their *reference pixel arrays*, not the full tiles. A reference pixel array is computed for each side of a tile by using a *pixel marching* (Fig. 5 (Left)), starting from the sides of the tile and collecting for each ray the colour value of the first encountered pixel that is occupied. When no pixel is found on a complete traversal, we use a black pixel in the associated pixel of the reference pixel array (see Fig. 5 (Right)). Once the four reference pixel arrays are generated for each tile (one per side, as in Fig. 5 (Right)), we can compare two tiles by any of their sides using MSE on those pixel arrays. When comparing one tile to two other tiles at the same time, as for step 4 in Fig. 4, we use the sum of the two MSEs.

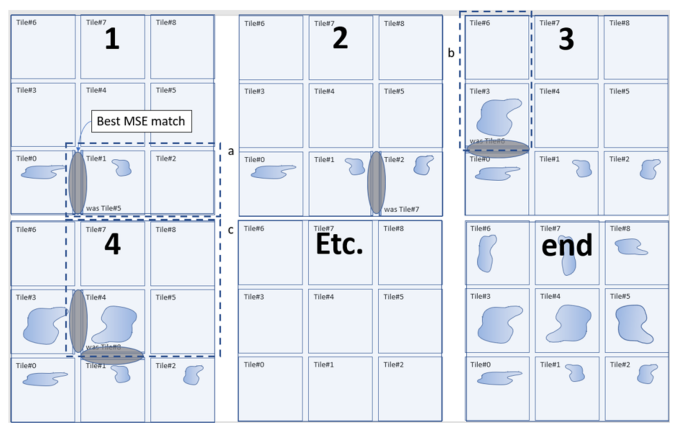

Fig. 4. Intra-frame texture tile reorganisation process.

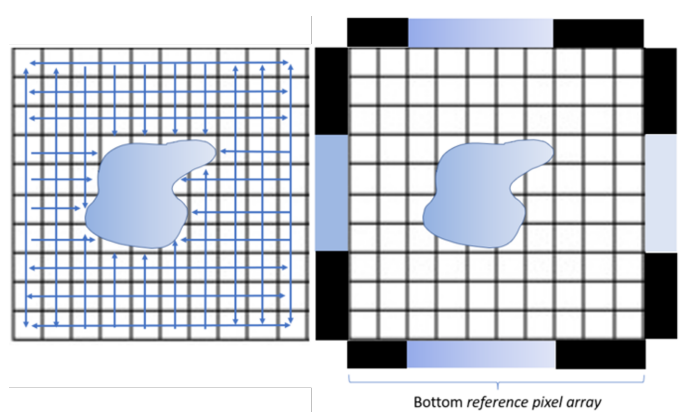

Fig. 5. Generating the *reference pixel arrays* (Right) for the four sides of a tile (Left) using simple pixel marching (Left).

Following the intra-frame tile reorganisation, we can also optionally apply an inter-frame tile reorganisation, to improve temporal correlations in the reprojected texture maps. The inter-frame reorganisation is applied in each GOP based on (and constrained by) the intra-frame reorganisation of the first frame in the same GOP. The tiles in each subsequent frame in the same GOP are compared using MSE with the tiles of the previous frame in the same GOP (this time computing MSE between *full* tiles, not between their reference pixel arrays), and the best-matching tile is moved to same location in the tile grid as its match in the previous frame. Rotations of tiles could also be applied during the MSE tests, to find the best-matching tile, but this has not yet been implemented. At the end of this process, we obtain a set of meshes whose topology is left unchanged, but the *uv* coordinates are changed to parameterize a set of stabilised tiled texture atlases that are more suitable for video-based compression than the original texture atlases. Fig. 1 (Right) illustrates an example of animated texture frames once stabilised with our solution and padded. The intra- and inter-frame tile reorganisation is tracked by a tile transform table, which is stored as metadata in the bitstream that is sent to the decoder.

5) Bitstream encoding: Whether or not the mesh segmentation and associated optional processes (described above) are applied, the mesh geometry and connectivity are encoded per frame using Draco [6] (we use a modified implementation, which skips the quantization operation). For the texture maps, these are first converted to the YUV colour space, then HDRTools [23] and the HM encoder [17] are used to encode the texture maps into an HEVC bitstream. *All-intra*, *random-access* and even lossless modes can be used for the HEVC encoding. The *uv* texture coordinates are only explicitly encoded if no mesh segmentation is used (see Section III-A1).

B. Decoder

Whether or not the input mesh is segmented into patches at the encoder, the texture maps are decoded using the HM decoder [17] and converted back into PNG image format using HDRTools [23]. For the decoding of the mesh geometry and connectivity, the first step is to apply the Draco decoder [6]. If no mesh segmentation was used at the encoder, this Draco decoding will directly produce the decoded mesh geometry, connectivity, and *uv* texture coordinates. If mesh segmentation was used, the Draco decoding will produce the geometry and connectivity only. In the latter case, the decoded mesh must then be segmented into patches and reparameterized into a new *uv* coordinate system in the same way as at the encoder (see Sections III-A2 and III-A3). If intra- and/or inter-frame texture tile reorganisation was performed at the encoder, at the decoder the tile reorganisation table from the received metadata must be used to transform the regenerated patch *uv* coordinates so that they correctly dereference the tiles that were moved during the reorganisation steps at the encoder. The decoding algorithm generates *uv* coordinates in floating point values, which can be optionally quantized (this was a requirement for the CfP [1]).

IV. Experimental Results

The experimental results that we present in this section are based on the mesh test data, error metrics, and test conditions described in the mesh coding CfP [1]. In Table I, we provide results for the lossy all-intra (AI) test condition ("C1" in [1]) and the lossy random-access (RA) test condition ("C2" in [1]). These results show the *Bjøntegaard-Delta* (BD) rate [24] between our proposed solution and the anchor codec used for the CfP [1], averaged across all the mesh sequences in each of the three dataset categories chosen for the CfP (explained in [1]). A negative BD-Rate percentage (green cells in Table I), e.g., $-X\%$, indicates that the proposed codec has, on average, $X\%$ *lower* bitrate than the anchor over the target bitrate range (see [1] for details), for the same mesh reconstruction quality. The reconstruction quality is measured as the *Peak Signal to Noise Ratio* (PSNR), using several different error metrics as indicated in the columns of Table I – see [1] for details. A positive BD-Rate percentage (pink cells in Table I) indicates a *higher* average bitrate than the anchor. We see that for both the mesh geometry (*D1*, *D2*, and *Geo.* columns) and the mesh colour (Luma (*Y*) and Chroma (*Cb* and *Cr*)), the proposed solution generally outperforms the anchor by a significant amount. Note that these results correspond to the version of the proposed codec where the mesh segmentation and related optional procedures (see Section III-A) have been applied. In Fig. 6, we also present some example R-D results for one of the mesh sequences used in the CfP [1], where our proposed solution is compared against the other (anonymous) proponents that responded to the CfP. In Fig. 6, our proposed solution is both "P14" and "P20". P14 corresponds to the case where the source for the texture attribute transfer in our encoder is the decimated mesh (see Section III-A3), and P20 is when the original input mesh is the source. Note that the results in Table I correspond to P20 averaged across different datasets. The results in Fig. 6 are representative of our results for most of the mesh sequences used for the CfP. Here we see that our proposed solution is generally in the top two of the submitted solutions, both for geometry and colour coding, and significantly outperforms the anchor codec for most of the target bitrates.

TABLE I
LOSSY GEOMETRY AND LOSSY COLOUR CODING RESULTS, USING TEST CONDITIONS: (TOP) C1: ALL-INTRA; (BOTTOM) C2: RANDOM-ACCESS

	Point cloud-based BD-Rate (%)					**Image-based BD-Rate (%)**	
C1 (AI)	***D1***	***D2***	***Y***	***Cb***	***Cr***	***Geo.***	***Y***
Cat1-A avg.	-40.8	-43.7	-15.6	-15.5	-14.5	-46.7	-20.8
Cat1-B avg.	-0.9	-16.3	-34.6	-41.0	-32.8	2.1	-32.4
Cat1-C avg.	-38.7	-38.3	-56.3	-69.9	-61.7	-41.8	-55.9
Overall avg.	**-29.8**	**-34.2**	**-40.7**	**-49.1**	**-42.7**	**-32.0**	**-41.3**
C2 (RA)							
Cat1-A avg.	-36.8	-39.8	-21.7	-26.9	-26.9	-43.2	-25.2
Cat1-B avg.	9.1	-14.0	-43.6	-50.5	-44.6	9.6	-37.5
Cat1-C avg.	-41.9	-42.0	-29.0	-1.2	2.7	-44.5	-34.2
Overall avg.	**-27.8**	**-34.5**	**-30.8**	**-19.9**	**-16.5**	**-30.7**	**-32.8**

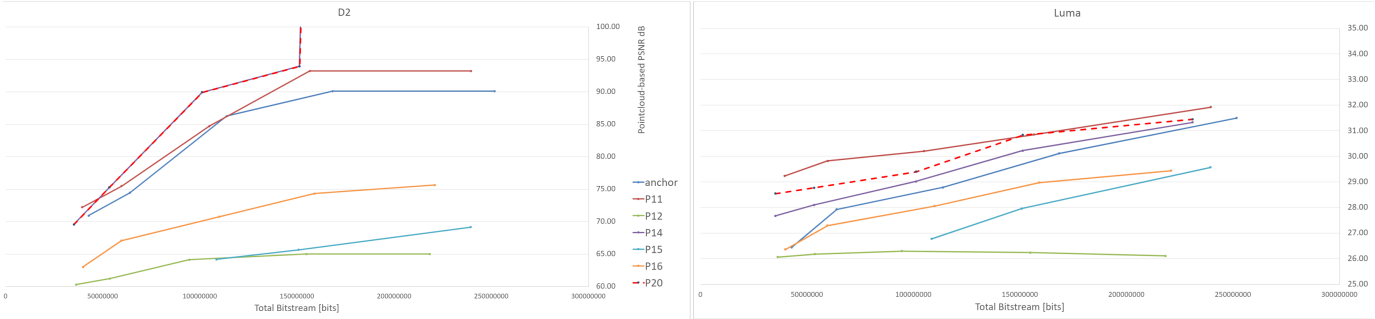

Fig. 6. Example R-D results for mesh sequence "Mitch" [1], for C1 (AI), using the point cloud-based PSNR (see [1]) for geometry (D2) and luma (Y in Table I). Our proposal is both "P14" and "P20".

V. Conclusion

In this paper, we presented a new method for compressing time-varying textured 3D mesh sequences, in response to the recent Call for Proposals [1] from the MPEG-3DGC group. Compared to the anchor codec used in the CfP [1], our proposal demonstrates notable rate-distortion performance improvements, for the all-intra and random-access test conditions, and for lossy geometry and lossy colour coding. Our solution also outperforms most other competing solutions that were submitted as responses to the CfP. Further improvements in R-D performance for our proposed method are still possible with better tuning and combinations of different coding parameters. In terms of algorithmic improvements, the key future work that we envision could involve: temporal patch stabilisation through temporally constrained mesh segmentation (which would lead to more efficient texture tile tracking), better inter-tile tracking by implementing tile rotations or other rigid transformations, and perhaps most importantly an extension of our algorithms to handle temporal mesh compression, which would result in significant gains in rate-distortion performance.

Acknowledgment

The authors would like to thank their colleagues at InterDigital and in the MPEG-3DGC group, for valuable discussions that led to the formulation of this work. We also thank the reviewers of this paper for their constructive feedback.

References

[1] ISO/IEC JTC 1/SC 29/WG 7, *CfP for Dynamic Mesh Coding*, Oct. 2021.

[2] ISO/IEC JTC 1/SC 29, *Information technology — Coding of audio-visual objects — Part 16: Animation Framework eXtension (AFX)*, 2011. [Online]. Available: https://www.iso.org/standard/57367.html?browse=tc

[3] H. Habe, Y. Katsura, and T. Matsuyama, "Skin-off: representation and compression scheme for 3d video," in *Picture Coding Symposium*, 2004, pp. 301–306.

[4] S.-R. Han, T. Yamasaki, and K. Aizawa, "3d video compression based on extended block matching algorithm," in *2006 International Conference on Image Processing*. IEEE, 2006, pp. 525–528.

[5] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 1997, pp. 209–216.

[6] Google, "Draco 3d data compression." [Online]. Available: https://google.github.io/draco/

[7] J. Rossignac, "Edgebreaker: Connectivity compression for triangle meshes," *IEEE transactions on visualization and computer graphics*, vol. 5, no. 1, pp. 47–61, 1999.

[8] J. Hou, L.-P. Chau, Y. He, and N. Magnenat-Thalmann, "A novel compression framework for 3d time-varying meshes," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2014, pp. 2161–2164.

[9] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h.264/avc video coding standard," *IEEE Transactions on circuits and systems for video technology*, vol. 13, no. 7, pp. 560–576, 2003.

[10] 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, 2007.

[11] S. R. Han, T. Yamasaki, and K. Aizawa, "Geometry compression for time-varying meshes using coarse and fine levels of quantization and run-length encoding," in *International Conference on Image Processing*, 11 2008, pp. 1045 – 1048.

[12] T. Yamasaki and K. Aizawa, "Patch-based compression for time-varying meshes," in *2010 IEEE International conference on image processing*. IEEE, 2010, pp. 3433–3436.

[13] L. Yamasaki and K. Aizawa, "Bit allocation of vertices and colors for patch-based coding in time-varying meshes," in *28th Picture Coding Symposium*, 2010, pp. 162–165.

[14] E. Faramarzi, R. Joshi, and M. Budagavi, "Mesh coding extensions to mpeg-i v-pcc," in *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*, 2020, pp. 1–5.

[15] D. B. Graziosi, "Video-based dynamic mesh coding," in *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2021, pp. 3133–3137.

[16] ISO/IEC JTC 1/SC 29, *Information technology — Coded representation of immersive media — Part 5: Visual volumetric video-based coding (V3C) and video-based point cloud compression (V-PCC)*, 2021. [Online]. Available: https://www.iso.org/standard/73025.html

[17] Z. M. Miličević and Z. S. Bojković, "Performance of high efficiency video coding (hevc) hm-16.6 encoder," in *2015 23rd Telecommunications Forum Telfor (TELFOR)*. IEEE, 2015, pp. 712–715.

[18] S. Shlafman, A. Tal, and S. Katz, "Metamorphosis of polyhedral surfaces using decomposition," in *Computer graphics forum*, vol. 21, no. 3. Wiley Online Library, 2002, pp. 219–228.

[19] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[20] R. Sawhney and K. Crane, "Boundary first flattening," *ACM Transactions on Graphics (ToG)*, vol. 37, no. 1, pp. 1–14, 2017.

[21] S. Yoshizawa, A. Belyaev, and H.-P. Seidel, "A fast and simple stretch-minimizing mesh parameterization," in *Proceedings Shape Modeling Applications, 2004*. IEEE, 2004, pp. 200–208.

[22] B. Lévy, S. Petitjean, N. Ray, and J. Maillot, "Least squares conformal maps for automatic texture atlas generation," *ACM transactions on graphics (TOG)*, vol. 21, no. 3, pp. 362–371, 2002.

[23] "Hdrtools." [Online]. Available: https://gitlab.com/standards/HDRTools

[24] G. Bjøntegaard, "Calculation of average psnr differences between rd-curves," *VCEG-M33*, 2001.