DYNAMIC POLYGON CLOUD COMPRESSION

Eduardo Pavez [1], *Philip A. Chou*[2], *Ricardo L. de Queiroz*[3]

[1]University of Southern California, Los Angeles, CA, USA
[2]Microsoft Research, Redmond, WA, USA
[3]Universidade de Brasilia, Brasilia, Brazil

ABSTRACT

We introduce compressible representation of 3D geometry (including its attributes, such as color texture) intermediate between polygonal meshes and point clouds called *polygon cloud*. Dynamic or time-varying polygon clouds, like dynamic polygonal meshes and dynamic point clouds, can take advantage of temporal redundancy for compression, if certain challenges are addressed. In this paper, we propose methods for compressing both static and dynamic polygon clouds, that apply transform coding on color and motion residuals. We find that triangle clouds can be compressed nearly as well as triangular meshes and point clouds. While being more robust to live capture noise and artifacts.

Index Terms— polygon cloud compression, transform coding, RAHT, predictive coding

1. INTRODUCTION

With the advent of virtual and augmented reality, live captured 3D content that can be experienced from any point of view. Such content ranges from static scans of compact 3D objects, to dynamic captures of non-rigid objects such as people, rooms, public spaces swarming with people, etc. For such content to be captured at one place and delivered to another for consumption by a virtual or augmented reality device (or by more conventional means), the content needs to be represented and compressed for transmission or storage. Applications include gaming, tele-immersive communication, live events, acquisition for special effects, etc. This paper presents a novel means of representing and compressing the visual part of such content.

Until this point, two of the more promising approaches to representing both static and time-varying 3D scenes have been polygonal meshes and point clouds, along with their associated color information. However, both approaches have drawbacks. Polygonal meshes represent surfaces very well, but they are not robust to noise and other structures typically found in live captures, such as lines, points, and ragged boundaries that violate the assumptions of a smooth surface manifold. Point clouds, on the other hand, have a hard time modeling surfaces as compactly as meshes.

We propose a hybrid between polygonal meshes and point clouds: polygon clouds. Polygon clouds are sets of polygons (possible overlapping) that are not required to represent a coherent surface. Like point clouds, a polygon cloud can represent noisy, real-world geometry captures without any assumption of a smooth 2D manifold. In fact, any polygon in a polygon cloud can be collapsed into a point or line as a special case. On the other hand, the polygons in the cloud can be stitched together into a watertight mesh to represent a smooth surface.

Compression of 3D meshes spans a long history in the computer graphics community [1, 2, 3]. Mesh compression involves coding connectivity [4, 5], vertex coordinates and surface color. One practical approach to compress geometry and color simultaneously is based on "geometry images" [6] and their temporal extension, "geometry videos" [7] for dynamic time varying meshes. In these approaches the mesh is projected onto a 2D image or video, which is compressed using a video coder.

A critical part of dynamic mesh compression is the ability to track points over time and producing time consistent frames, i.e. each point in the mesh can be traced from one frame to the next. We are particularly influenced by [8, 9, 10], all of which produce in real time, given data from one or more RGBD sensors for every frame, a parameterized mapping that maps points in consecutive frames.

Point cloud compression has shown more robustness for real time capture and display of 3D geometry, however it is more challenging. Sparse Voxel Octtres (SVOs) were developed to represent geometry of 3D objects [11, 12]. Octrees were first used for point cloud compression in [13] and for color attribute compression, in [14].

For static voxelized point clouds the state of the art methods in color compression is the system developed by [15] which is based in the Graph Fourier Transform (GFT), however it is computationally expensive for real time applications.

E. Pavez is with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA, e-mail: pavezcar@usc.edu

P. A. Chou is with Microsoft Research, Redmond, WA, USA, e-mail: pachou@ieee.org.

R. L. de Queiroz is with the Computer Science Department at Universidade de Brasilia, Brasilia, Brazil, e-mail: queiroz@ieee.org.

A more practical approach is the Region Adaptive Hierarchical Transform (RAHT) [16], which has similar performance and can be implemented efficiently in a GPU.

In [17] a system for dynamic voxelized point clouds was proposed. They find matches between consecutive frames and use them for prediction of color attributes, then the residuals are passed through a transform coding system based in the GFT. Other works that use motion estimation and compensation to encode color residuals are [18, 19].

This paper is organized as follows, in section 2 we introduce polygon clouds and our compression system. In section 3 we discuss the compression systems in more detail. We show color and geometry compression results in section 4 and end with conclusions in 5.

2. SYSTEM OVERVIEW

2.1. Notation

We denote scalars a, vectors and matrices with bold font as $\mathbf{a}$, $\mathbf{A}$. Caligraphic letters denote sets. And we define the set of integers between 1 and N by $[N] = \{1, \cdots, N\}$. We denote by $\|\cdot\|_2$ the ℓ_2 norm of a vector and by $\|\cdot\|_F$ the Frobenius norm of a matrix.

2.2. Dynamic triangle clouds

A dynamic triangle cloud is a numerical representation of a time changing 3D scene or object. We denote it by a sequence $\{\mathcal{T}^{(t)}\}$ where $\mathcal{T}^{(t)}$ is a triangle cloud at time t. Each individual frame $\mathcal{T}^{(t)}$ has geometry (shape and position) and color information.

The geometry information consists of a list of vertices $\mathcal{V}^{(t)} = \{v_i^{(t)} : i = 1, \cdots, N_p\}$, where each vertex $v_i^{(t)} = [x_i^{(t)}, y_i^{(t)}, z_i^{(t)}]$ is a point in 3D, and a list of triangles (or faces) $\mathcal{F}^{(t)} = \{f_m^{(t)} : m = 1, \cdots, N_f\}$, where each face $f_m^{(t)} = [i_m^{(t)}, j_m^{(t)}, k_m^{(t)}]$ is a vector of indices of vertices from $\mathcal{V}^{(t)}$. We denote by $\mathbf{V}^{(t)}$ the $N_p \times 3$ matrix whose ith row is the point $v_i^{(t)}$, and similarly we denote by $\mathbf{F}^{(t)}$ the $N_f \times 3$ matrix whose mth row is the triangle $f_m^{(t)}$. The triangles in a triangle cloud do not have to be adjacent or form a mesh, and they can overlap. Two or more vertices of a triangle may have the same coordinates, thus collapsing into a line or point.

The color information consists of a list $\mathcal{C}^{(t)} = \{c_n^{(t)} : n = 1, \cdots, N_c\}$, where each color $c_n^{(t)} = [Y_n^{(t)}, U_n^{(t)}, V_n^{(t)}]$ is a vector in YUV space (or other convenient color space). We denote by $\mathbf{C}^{(t)}$ the $N_c \times 3$ matrix whose n-th row is the color $c_n^{(t)}$. The list the colors across the surfaces of the triangles. To be specific, $c_n^{(t)}$ is the color of a "refined" vertex $v_r^{(t)}(n)$, where the refined vertices are obtained by uniformly subdividing each triangle in $\mathcal{F}^{(t)}$ by upsampling factor U, as shown in Figure 1 for $U = 4$. We denote by $\mathbf{V}_r^{(t)}$ the $N_c \times 3$ matrix whose nth row is the refined vertex $v_r^{(t)}(n)$. $\mathbf{V}_r^{(t)}$ can be computed from $\mathcal{V}^{(t)}$ and $\mathcal{F}^{(t)}$, so we do not need to encode it, but we will use it to compress the color information. Note that $N_c = N_f(U+1)(U+2)/2$. The upsampling factor U should be high enough so that it does not limit the color spatial resolution obtainable by the color cameras. In our experiments, we set $U = 10$ or higher. Setting U higher does not typically affect the bit rate significantly, though it does affect memory and computation in the encoder and decoder.

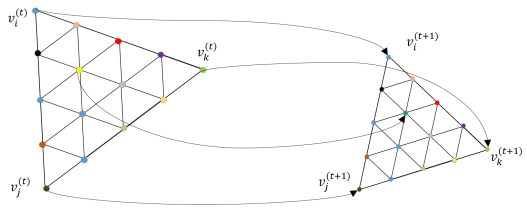


Fig. 1: Correspondences between two consecutive frames.

Thus frame t can be represented by the triple $\mathbf{V}^{(t)}$, $\mathbf{F}^{(t)}$, $\mathbf{C}^{(t)}$. We use a Group of Frames (GOF) model, in which the sequence is partitioned into GOFs. Without loss of generality, we label the frames in a GOF $t = 1 \ldots, N$. There are two types of frames: reference and predicted. In each GOF, the first frame ($t = 1$) is a reference frame and all other frames ($t = 2, \ldots, N$) are predicted. Within a GOF, all frames must have the same number of vertices, triangles, and colors: $\forall t \in [N]$, $\mathbf{V}^{(t)} \in \mathbb{R}^{N_p \times 3}$, $\mathbf{F}^{(t)} \in [N_p]^{N_f \times 3}$ and $\mathbf{C}^{(t)} \in \mathbb{R}^{N_c \times 3}$. The triangles are assumed to be consistent across frames so that there is a correspondence between colors and vertices within the GOF. In Figure 1 we show an example of the correspondences between two consecutive frames in a GOF. Different GOFs may have a different numbers of frames, vertices, triangles, and colors.

2.3. Compression of dynamic triangle clouds

In this section we provide an overview of our system for compressing dynamic triangle clouds. We compress consecutive GOFs sequentially and independently, so we focus on the system for compressing an individual GOF $(\mathbf{V}^{(t)}, \mathbf{F}^{(t)}, \mathbf{C}^{(t)})$ for $t \in [N]$.

2.3.1. Reference frames

We encode the connectivity $\mathbf{F}^{(1)}$ with a lossless entropy coder *gzip*.

The vertices $\mathbf{V}^{(1)}$ are converted to voxels $\mathbf{V}_v^{(1)}$. While converting to voxels, many points may fall into the same voxel, thus we remove repeated vertices when creating $\mathbf{V}_v^{(1)}$. We keep track of removed points using a list of flags $\mathbf{I}_v$. The voxels are represented using and octtree which is compressed using *gzip*. The list $\mathbf{I}_v$ is also coded with *gzip*. The decompressed vertices $\hat{\mathbf{V}}^{(1)}$can be recovered by inverting the octtree scanning and by using the indices. To compress color we use

our triangle refinement function on the decoded vertices and produce $\hat{\mathbf{V}}_r^{(1)}$. Each point in this set of refined vertices can be mapped to an unique color in the matrix $\mathbf{C}^{(1)}$, thus we have a *static* point cloud $(\hat{\mathbf{V}}_r^{(1)}, \mathbf{C}^{(1)})$. The color attributes of this point cloud are voxelized and compressed using a transform coding system composed of the RAHT [16] followed by uniform scalar quantization and Run-Length-Golomb-Rice (RLGR)[22] entropy coder. We will discuss this transform coding system in more detail in the next section.

2.3.2. Predicted frames

Since triangles do not change within a GOP, i.e. $\mathbf{F}^{(t)} = \mathbf{F}^{(1)}$, we only need to compress vertices and color. We compute prediction residuals from the previously decoded frame. Specifically, for each predicted frame $t > 1$ we compute a motion residual $\Delta\mathbf{V}^{(t)} = \mathbf{V}^{(t)} - \hat{\mathbf{V}}^{(t-1)}$ and a color residual $\Delta\mathbf{C}^{(t)} = \mathbf{C}^{(t)} - \hat{\mathbf{C}}^{(t-1)}$, where we have denoted with a *hat* a quantity that has been compressed and decompressed. The motion residuals and color residuals are considered as attributes of a point cloud in the reference frame. Specifically, the point clouds we use are defined as $(\hat{\mathbf{V}}^{(1)}, \Delta\mathbf{V}^{(t)})$ with motion residual as attributes, and $(\hat{\mathbf{V}}_r^{(1)}, \Delta\mathbf{C}^{(t)})$ with color residuals as attributes. These attributes are voxelized and compressed using the transform coding system based on RAHT followed by uniform scalar quantization and RLGR encoding. The voxelization process ensures that 1) if two or more vertices or colors fall into the same voxel, they receive the same representation and hence are encoded only once, and 2) the colors (on the set of refined vertices) are resampled uniformly in space regardless of the density and size of triangles.

In the next section, we describe the basic elements of the system: voxelization, octrees, and transform coding.

3. ATTRIBUTE VOXELIZATION, OCTTREE AND TRANSFORM CODING

3.1. Morton codes and voxelization

A *voxel* is a volumetric element used to represent the attributes of of an object in 3D over a small region of space. Analogous to 2D pixels, 3D voxels are defined on a uniform grid. We assume the geometric data live in the unit cube $[0,1)^3$, and we uniformly partition the cube into voxels of size $2^{-J} \times 2^{-J} \times 2^{-J}$.

Now consider a point cloud with points $\mathbf{V} = [v_i]$ and their corresponding attributes $\mathbf{A} = [a_i]$, where a_i is the real-valued attribute (or vector of attributes) of v_i. (These may be, for example, the list of refined vertices $\hat{\mathbf{V}}_r$ and their associated colors $\mathbf{C}$ or color residuals as discussed above.) In the process of *voxelization*, the points are partitioned into voxels, and the attributes associated with the points in a voxel are averaged. The points within each voxel are quantized to the voxel center. Each occupied voxel is then represented by the voxel center and the average of the attributes of the points in the voxel. Moreover, the occupied voxels are put into Z-scan order, also known as Morton order [23]. The first step in voxelization is to quantize the vertices and to produce their Morton codes. The Morton code m for a point (x, y, z) is obtained simply by interleaving the bits of x, y, and z, with x being lower order than y, and y being lower order than z. The Morton codes are sorted, duplicates are removed, and all attributes whose vertices have the same Morton code are averaged.

$\mathbf{V}_{int}$ is the list of vertices with their coordinates, previously in $[0,1)$, now mapped to integers in $\{0, \ldots, 2^J - 1\}$. $\mathbf{M}$ is the corresponding list of Morton codes. $\mathbf{M}_v$ is the list of Morton codes, sorted with duplicates removed, using the Matlab function *unique*. $\mathbf{I}$ is a vector of indices such that $\mathbf{M}_v = \mathbf{M}(\mathbf{I})$ and $\mathbf{A}_v = [\bar{a}_j]$ is the list of attribute averages

$$\bar{a}_j = \frac{1}{N_j} \sum_{i:\mathbf{M}(i)=\mathbf{M}_v(j)} a_i, \tag{1}$$

where N_j is the number of elements in the sum. $\mathbf{V}_v$ is the list of voxel centers. The algorithm has complexity $\mathcal{O}(N \log N)$, where N is the number of input vertices.

3.2. Octree encoding

Any set of voxels in the unit cube, each of size $2^{-J} \times 2^{-J} \times 2^{-J}$, designated *occupied* voxels, can be represented with an octree of depth J [11, 12]. An octree is a recursive subdivision of a cube into smaller cubes, as illustrated in Figure 2. Cubes are subdivided only as long as they are occupied (i.e., contain any occupied voxels). This recursive subdivision can be represented by an octree with depth J, where the root corresponds to the unit cube. The leaves of the tree correspond to the set of occupied voxels.

There is a close connection between octrees and Morton codes. In fact, the Morton code of a voxel, which has length $3J$ bits broken into J binary triples, encodes the path in the octree from the root to the leaf containing the voxel. Moreover, the sorted list of Morton codes results from a depth-first traversal of the tree.

Each internal node of the tree can be represented by one byte, to indicate which of its eight children are occupied. If these bytes are serialized in a pre-order traversal of the tree, the serialization (which has a length in bytes equal to the number of internal nodes of the tree) can be used as a description of the octree, from which the octree can be reconstructed. Hence the description can also be used to encode the ordered list of Morton codes of the leaves. For simplicity in our experiments, we use *gzip* instead of an arithmetic encoder. In this way, we encode any set of occupied voxels in a canonical (Morton) order.

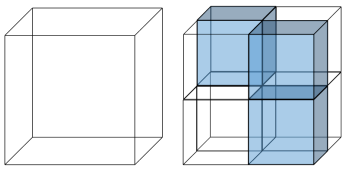


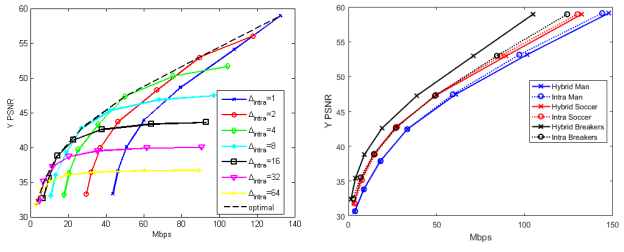
Fig. 2: Cube subdivision. Blue cubes represent occupied regions of space.

3.3. Transform coding

The RAHT [16] can be described as a sequence of orthonormal transforms applied to attribute data living on the leaves of an octree. For simplicity we assume the attributes are scalars. This transform processes voxelized attributes in a bottom up fashion, starting at the leaves of the octree. The inverse transform reverses this order.

The RAHT can be interpreted as a weighted Haar transform that processes pairs of attributes living in the nodes of a octtree. If two attributes have the same parent, they will be processed with a weighted average and a difference. After applying the RAHT, there remains one low pass coefficient that corresponds to the DC component; the remainder are high pass coefficients. Since after each processing of a pair of coefficients, the weights are added and used during the next transformation, the weights can be interpreted as being inversely proportional to frequency. The DC coefficient is the one that has the largest weight, as it is processed more times and represents information from the entire cube, while the high pass coefficients, which are produced earlier, have smaller weights because they contain information from a smaller region. The weights depend only on the octree (not the coefficients themselves), and thus can provide a frequency ordering for the coefficients. We sort the transformed coefficients by decreasing magnitude of weight.

Finally, the sorted coefficients are quantized using uniform scalar quantization, and are entropy coded using adaptive Run Length Golomb-Rice coding [22].



(a) RD *soccer* (b) Hybrid vs Intra

Fig. 3: RD curves for color compression.

4. EXPERIMENTS

We analyze the RD performance of our compression system with the Microsoft HCap *man, soccer, breakers* sequences. Each frame is a triangular mesh, within a group of frames, the meshes are consistent, i.e. the connectivity is fixed but the locations of the triangle vertices evolve in time. The geometry is a list of triangles and the xyz coordinates of their vertices, for frame the color information is stored in an image atlas and each triangle can be mapped to a triangular region of the image for rendering. To construct a triangle cloud we sample the image atlas using our refinement fuction with an upsample factor of $U = 10$. All sequences have 30 frames per second.

We consider a fixed voxel size of $2^{-J}W \times 2^{-J}W \times 2^{-J}W$ where W is the size of the bounding box of a sequence. $J = 10$ is the maximum depth of the octtree.

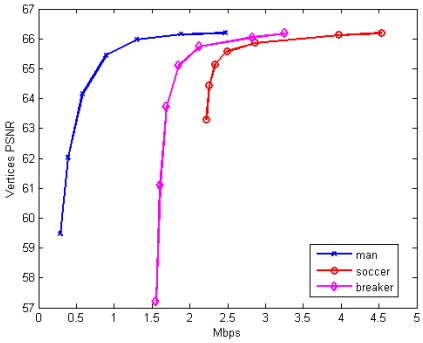
4.1. Color

For color coding we consider separate quantization steps for reference and predicted frames. The parameters take values $\Delta_{color,intra}, \Delta_{color,inter} \in \{1, 2, 4, 8, 16, 32, 64\}$. To analyze the coding performance, we report peak signal-to-noise ratio for the voxelized Y color component before and after TC (RAHT and quantization) as $psnr = -10 \log_{10} \left(\frac{1}{T} \sum_{t=1}^{T} \frac{1}{255^2 N_v^{(t)}} \| \mathbf{C}_v^{(t)} - \hat{\mathbf{C}}_v^{(t)} \|_2^2 \right)$, and report bit rate for all color components(YUV). In figure 3a we show PSNR (Y) vs Rate of color compression (YUV) for different combinations of quantization steps for intra and inter for the *soccer* sequence. We observe that the optimal RD curve is obtained by choosing $\Delta_{color,intra} = \Delta_{color,inter}$ as shown in the dashed line. This observation is also true for *man* and *breaker* sequence but not shown for space limitations. Now we compare against a system where all frames ere encoded in intra mode. We show the RD plots in figure 3b for all three sequences. We observe that the proposed hybrid (Intra-Inter) system outperforms the all intra system only for the *breaker* sequence. This suggest further investigation in when and how we can achieve gains with predictive coding.

4.2. Geometry

Since we the geometry for reference frames using octtree and lossless methods for connectivity and indices of collapsed vertices, we only analyze compression on predicted frames. For that we compute PSNR for the vertex xyz coordinates before and after transform coding using $psnr = -10 \log_{10} \left(3\epsilon^2 + \frac{1}{T_{inter}} \sum_{t \text{ is inter}} \frac{\| \mathbf{V}_v^{(t)} - \hat{\mathbf{V}}_v^{(t)} \|_F^2}{W^2 N_v^{(t)}} \right)$, where $\epsilon^2 = \frac{1}{12}(W/2^J)^2$ is the quantization error introduced by putting vertices into voxels. For bit rate we add the number of bits required for coding of vertices, indices and faces. We observe in figure 4a that motion artifacts are less visible for PSNR values above $65db$, which can be achieved at very

small rates. The encoder in all intra mode which uses octtree for all frames has rate of $5.2447[Mbps], 6.3889[Mbps]$ and $4.8797[Mbps]$ for *man, soccer, breaker* sequences respectively. In figures 4b-d we show the effect of motion compression for *breaker* sequence PSNR and can see that by increasing the bit rate slightly, the psnr goes from $57db$ to $65db$, which translates into a smooth motion.



(a) RD curves for motion compression.

(b) original (c) $2.13Mbps$ (d) $1.55Mbps$

Fig. 4: Geometry compression.

5. CONCLUSION

6. REFERENCES

[1] P. Alliez and C. Gotsman, "Recent advances in compression of 3d meshes," in *Advances in Multiresolution for Geometric Modeling*, N. A. Dodgson, M. S. Floater, and M. A. Sabin, Eds., pp. 3–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

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

[3] A. Maglo, G. Lavoué, F. Dupont, and C. Hudelot, "3d mesh compression: survey, comparisons and emerging trends," *ACM Computing Surveys*, vol. 9, no. 4, 2013.

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

[5] K. Mamou, T. Zaharia, and F. Prêteux, "TFAN: A low complexity 3d mesh compression algorithm," *Computer Animation and Virtual Worlds*, vol. 20, 2009.

[6] Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe, "Geometry images," *ACM Trans. Graphics (SIGGRAPH)*, vol. 21, no. 3, pp. 355–361, July 2002.

[7] H. Briceño, P. Sander, L. McMillan, S. Gortler, and H. Hoppe, "Geometry videos: a new representation for 3d animations," in *Symp. Computer Animation*, 2003.

[8] R. A. Newcombe, D. Fox, and S. M. Seitz, "Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 343–352.

[9] M. Dou, J. Taylor, H. Fuchs, A. Fitzgibbon, and S. Izadi, "3d scanning deformable objects with a single rgbd sensor," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 493–501.

[10] M. Dou, S. Khamis, Y. Degtyarev, P. Davidson, S. R. Fanello, A. Kowdle, S. Orts Escolano, C. Rhemann, D. Kim, J. Taylor, P. Kohli, V. Tankovich, and S. Izadi, "Fusion4d: real-time performance capture of challenging scenes," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, pp. 114, 2016.

[11] C. L. Jackins and S. L. Tanimoto, "Oct-trees and their use in representing three-dimensional objects," *Computer Graphics and Image Processing*, vol. 14, no. 3, pp. 249 – 270, 1980.

[12] D. Meagher, "Geometric modeling using octree encoding," *Computer Graphics and Image Processing*, vol. 19, no. 2, pp. 129 – 147, 1982.

[13] R. Schnabel and R. Klein, "Octree-based point-cloud compression," in *Eurographics Symp. on Point-Based Graphics*, July 2006.

[14] Y. Huang, J. Peng, C. C. J. Kuo, and M. Gopi, "A generic scheme for progressive point cloud coding.," *IEEE Trans. Vis. Comput. Graph.*, vol. 14, no. 2, pp. 440–453, 2008.

[15] C. Zhang, D. Florêncio, and C. Loop, "Point cloud attribute compression with graph transform," in *2014 IEEE International Conference on Image Processing (ICIP)*, Oct 2014, pp. 2066–2070.

[16] R. L. de Queiroz and P. A. Chou, "Compression of 3d point clouds using a region-adaptive hierarchical transform," *IEEE Transactions on Image Processing*, vol. 25, no. 8, pp. 3947–3956, Aug 2016.

[17] 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.

[18] R. L. de Queiroz and P. A. Chou, "Motion-compensated compression of dynamic voxelized point clouds," *IEEE Trans. Image Processing*, 2016, submitted.

[19] 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. PP, no. 99, pp. 1–1, 2016.

[20] H. Q. Nguyen, P. A. Chou, and Y. Chen, "Compression of human body sequences using graph wavelet filter banks," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2014, pp. 6152–6156.

[21] A. Anis, P. A. Chou, and A. Ortega, "Compression of dynamic 3d point clouds using subdivisional meshes and graph wavelet transforms," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2016, pp. 6360–6364.

[22] H. S. Malvar, "Adaptive run-length/golomb-rice encoding of quantized generalized gaussian sources with unknown statistics," in *Data Compression Conference (DCC'06)*, March 2006, pp. 23–32.

[23] G. M Morton, "A computer oriented geodetic data base; and a new technique in file sequencing," Technical report, IBM, Ottawa, Canada, 1966.