DYNAMIC POLYGON CLOUD COMPRESSION

Eduardo Pavez [1] *and Philip A. Chou*[2]

[1]University of Southern California, Los Angeles, CA, USA
[2]Microsoft Research, Redmond, WA, USA

ABSTRACT

We introduce a compressible representation of 3D geometry (including its attributes, such as color texture) intermediate between polygonal meshes and point clouds called a *polygon cloud. Polygon clouds*, compared to polygonal meshes, are more robust to live capture noise and artifacts. Furthermore, *dynamic polygon clouds*, compared to dynamic point clouds, are easier to compress, if certain challenges are addressed. In this paper, we propose methods for compressing *dynamic polygon clouds* using transform coding of color and motion residuals. We find that, compared to static polygon clouds and a fortiori static point clouds, dynamic polygon clouds can improve color compression by up to 2-3 dB in fidelity, and can improve geometry compression up to a factor of 2-5 in bit rate.

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

1. INTRODUCTION

With the advent of virtual and augmented reality, live captured 3D content 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 and even whole cities in motion. 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, 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 (possibly 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 mesh to represent a smooth surface.

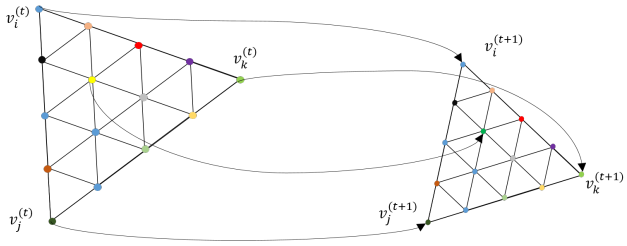


Fig. 1: Correspondences between two consecutive frames.

Compression of 3D meshes spans a long history in the computer graphics community [1, 2, 3, 4]. Mesh compression involves coding connectivity [5, 6], vertex coordinates, and surface color. One practical approach for compressing geometry and color simultaneously is based on "geometry images" [7] and their temporal extension, "geometry videos" [8] 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.

Point cloud compression has shown more robustness for real time capture and display of 3D geometry; however it is more challenging. Sparse Voxel Octrees (SVOs) were developed to represent geometry of 3D objects [9, 10]. Octrees were first used for point cloud compression in [11] and color attribute compression in [12].

For static voxelized point clouds a state of the art method for color compression is the system developed by [13], which is based on the Graph Fourier Transform (GFT), however it is computationally expensive for real time applications. A more practical approach is the Region Adaptive Hierarchical Transform (RAHT) [14], which has similar performance and can be implemented efficiently in a GPU. In [15] a system for dynamic voxelized point clouds was proposed. It finds matches between consecutive frames and uses them for prediction of color attributes, then it passes the residuals through a transform coding system based on the GFT. Other works that also use motion estimation and compensation to encode color residuals are [16, 17].

A critical element for efficient compression of time varying 3D scenes is tracking points over time and producing time-consistent frames, i.e., tracing each point (of the mesh, point cloud, or polygon cloud) from one frame to the next. We assume time consistent dynamic polygon clouds can be constructed in real time and we focus on real time compression. We are particularly influenced by [18, 19, 20], 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.

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 ge-

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 was with Microsoft Research, Redmond, WA, USA. He is now with 8i Labs, Inc., Bellevue, WA, USA, e-mail: pachou@ieee.org.

ometry compression results in Section 4 and conclude in Section 5.

2. SYSTEM OVERVIEW

2.1. Dynamic triangle clouds

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

The geometry information of a triangle cloud 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 i-th row is the point $v_i^{(t)}$, and similarly we denote by $\mathbf{F}^{(t)}$ the $N_f \times 3$ matrix whose m-th 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 of a triangle cloud consists of a list of colors $\mathcal{C}^{(t)} = \{c_n^{(t)} : n = 1, \cdots, N_c\}$, 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 $c_n^{(t)}$. $c_n^{(t)}$ is the color of a "refined" vertex $v_r^{(t)}(n)$, which is 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 n-th 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.

Each triangle cloud $\mathcal{T}^{(t)}$ can therefore be represented by a triplet $\mathbf{V}^{(t)}, \mathbf{F}^{(t)}, \mathbf{C}^{(t)}$. 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 $t = 1 \ldots, N$. In each GOF, the first frame ($t = 1$) is a reference frame and all other frames ($t = 2, \ldots, N$) are predicted frames. Within a GOF, all frames must have the same number of vertices, triangles, and colors: $\forall t \in \{1, \cdots, N\}$, $\mathbf{V}^{(t)} \in \mathbb{R}^{N_p \times 3}$, $\mathbf{F}^{(t)} \in \{1, \cdots, N_p\}^{N_f \times 3}$ 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.

2.2. 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 $\{(\mathbf{V}^{(t)}, \mathbf{F}^{(t)}, \mathbf{C}^{(t)}) : t \in \{1, \cdots, N\}\}$.

2.2.1. *Reference frames*

The vertices $\mathbf{V}^{(1)}$ are converted to voxels $\mathbf{V}_v^{(1)}$. During this process, some 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 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. The voxels are represented using an octree, which is further compressed using *gzip*. The list $\mathbf{I}_v$, as well as the connectivity $\mathbf{F}^{(1)}$, are also coded with *gzip*. The decompressed vertices $\hat{\mathbf{V}}^{(1)}$can be recovered by inverting the octree scanning and by using the indices. To compress color we use triangle refinement on the decoded vertices to produce $\hat{\mathbf{V}}_r^{(1)}$. Each point in this set of refined vertices can be mapped to a 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 region adaptive hierarchical transform (RAHT) [14] followed by uniform scalar quantization and Run-Length-Golomb-Rice (RLGR) [21] entropy coding. We will discuss this transform coding system in more detail in the next section.

2.2.2. *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 residuals 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.

3. OCTREE, ATTRIBUTE VOXELIZATION, AND TRANSFORM CODING

In this section we describe the basic elements of our compression system, which uses *voxels* to compress geometry and color.

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 lives in the unit cube $[0, 1)^3$, and we uniformly partition the cube into voxels of size $2^{-J} \times 2^{-J} \times 2^{-J}$.

For geometry compression in reference frames we use octree encoding. For color compression in reference and predicted frames, as well as for geometry of predicted frames we use a point cloud transform coding system consisting of a voxelization step followed by RAHT, uniform scalar quantization, and entropy coding.

3.1. Octree encoding

Octree encoding is an efficient technique used to represent a set of voxels. We use it to represent the vertices of the reference frames. We achieve that by quantizing points $\mathbf{V}^{(1)}$ to the center of a voxel in

the unit cube. A voxel is said to be *occupied* if there is a point that falls inside it. An octree is constructed by a recursive subdivision of the unit 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 [9, 10], where the root corresponds to the unit cube. The leaves of the tree correspond to the set of occupied voxels.

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 depth-first 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. To compress this representation further we use *gzip* in our experiments.

3.2. Voxelization and Morton codes

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 in Section 2). In this section we describe how to voxelize a point cloud, which is required by RAHT.

In the process of *voxelization* of a point cloud, the points v_i are assigned to voxels, and the attributes associated with the points in the same voxel are averaged. The points within each voxel are quantized to the voxel center. Each occupied voxel is represented by the voxel center and the average of the attributes of the points in the voxel.

An efficient way of implementing this process is by arranging the occupied voxels into Z-scan order, also known as Morton order [22]. Thus the first step in voxelization is to quantize the vertices and to produce their Morton codes. The Morton code m for a point $v = (x, y, z)$ is obtained simply by interleaving the bits of x, y, and z, with x being higher order than y, and y being higher order than z. The Morton codes are sorted, duplicates are removed, and all attributes whose vertices have the same Morton code are averaged.

If $\mathbf{M}$ is the corresponding list of Morton codes of points $\mathbf{V}$, then $\mathbf{M}_v$ is the list of Morton codes, sorted in increasing order 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 point cloud $(\mathbf{V}_v, \mathbf{A}_v)$ has been voxelized and can be compressed using RAHT. The quantization of vertex coordinates and the averaging of the attributes introduces distortion, which depends only on the density of the point cloud and the voxel size (controlled by J). The voxelization can be inverted by using the indices $\mathbf{I}$ to restore the ordering, and approximating the vertex coordinates and attributes by their voxelized values. The voxelization algorithm has complexity $\mathcal{O}(N \log N)$, where N is the number of input vertices.

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. Hence octree encoding described in the previous subsection is equivalent to coding the sorted list of Morton codes.

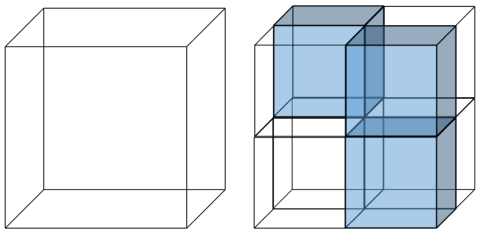


Fig. 2: Cube subdivision. Blue cubes represent occupied regions of space.

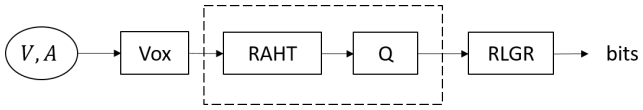


Fig. 3: Voxelization and transform coding diagram.

Furthermore, putting the voxelized point cloud (vertices and attributes) in Morton order can be exploited to implement RAHT efficiently.

3.3. Transform, quantization and entropy coding.

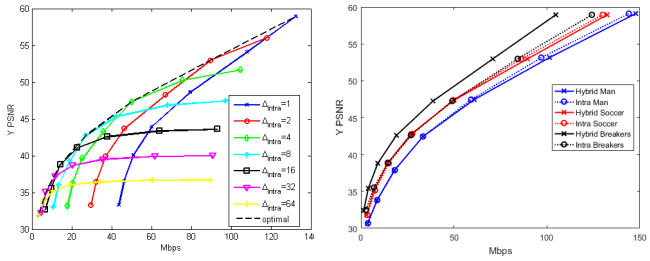
The system to compress the attributes of a voxelized point cloud is shown in Figure 3. RAHT [14] is a sequence of orthonormal transforms applied to attribute data living on the leaves of an octree. For simplicity we assume the attributes are scalars. This transform processes voxelized attributes in a bottom up fashion, starting at the leaves of the octree. The inverse transform reverses this order.

RAHT can be interpreted as a weighted Haar transform that processes attributes hierarchically according to the octree. If attributes have the same parent in the octree, they will be transformed into a weighted average and weighted differences. The weighted averages are low resolution attributes themselves and are transformed up the tree. At the end of the process, there is one low pass coefficient that corresponds to the DC component; the remainder are high pass coefficients. Since at each stage, the weights corresponding to processed attributes are added and used during the next stage, they 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 with bin size Δ, then entropy coded using adaptive Run Length Golomb-Rice coding [21].

4. EXPERIMENTS

We evaluate the RD performance of our system, for both color and geometry, comparing our GOF-based (hybrid intra-inter) system with an all-intra system. As data, we use triangle cloud sequences derived from the Microsoft HoloLens Capture (HCap) mesh sequences *man*, *soccer*, and *breakers*. In the HCap sequences, each frame is a triangular mesh, and 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. We construct a triangle cloud from



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

Fig. 4: RD curves for color compression.

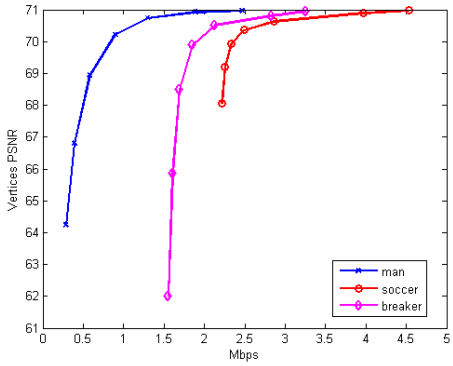
each mesh as follows. For the vertex and face lists, we use the vertex and face lists directly from the mesh. For the color list, we upsample each face by factor $U = 10$ to create a list of refined vertices, and then sample the mesh's texture map at the refined vertices. The geometric data are scaled to fit in the unit cube $[0, 1]^3$. Our voxel size is $2^{-J} \times 2^{-J} \times 2^{-J}$, where $J = 10$ is the maximum depth of the octree. All sequences are 30 frames per second.

4.1. Color

To evaluate color coding, we first consider separate quantization stepsizes for reference and predicted frames. The stepsizes take values $\Delta_{color,intra}$, $\Delta_{color,inter} \in \{1, 2, 4, 8, 16, 32, 64\}$. We report peak signal-to-noise ratio for the voxelized Y color component before and after transform coding (RAHT and quantization) as Y-PSNR $= -10 \log_{10} \left(\frac{1}{T} \sum_{t=1}^{T} \frac{1}{255^2 N_v^{(t)}} \|\mathbf{Y}_v^{(t)} - \hat{\mathbf{Y}}_v^{(t)}\|_2^2 \right)$, where T is the total number of frames in the sequence, $N_v^{(t)}$ is the number of occupied voxels in frame t, and $\mathbf{Y}_v^{(t)}$, $\hat{\mathbf{Y}}_v^{(t)}$ are the first columns of the matrices $\mathbf{C}_v^{(t)}$, $\hat{\mathbf{C}}_v^{(t)}$. We report bit rate for all color components (YUV). In Figure 4a we show Y-PSNR vs. color bit rate 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* sequences but is not shown due to space limitations. Now for equal step sizes, we compare our GOF-based hybrid intra-inter system with a system in which all frames ere encoded in intra mode. We show the RD plots in Figure 4b for all three sequences. We observe that the proposed hybrid intra-inter system outperforms the all-intra system by 2-3 dB for the *breakers* sequence. However, for the *man* and *soccer* sequences, their RD performances are similar. Further investigation is needed on when and how gains can be achieved by predictive coding of color.

4.2. Geometry

For reference frames the geometry is voxelized, and then losslessly entropy coded (using octrees plus *gzip* for vertices and *gzip* for connectivity and indices). Transform coding is applied only to predicted frames; thus we report PSNR for the vertex xyz coordinates before and after transform coding using $PSNR = -10 \log_{10} \left(\epsilon^2 + \frac{1}{T_{inter}} \sum_{t \text{ is inter}} \frac{\|\mathbf{V}_v^{(t)} - \hat{\mathbf{V}}_v^{(t)}\|_F^2}{3N_v^{(t)}} \right)$, where $\epsilon^2 = 2^{-2J}/12$ is the mean quantization error introduced by voxelization of reference frames that propagates to the predicted frames. T_{inter} is the number of predicted frames and $\|\cdot\|_F$ is the Frobenius norm. For bit rate we report the number of bits required for coding of vertices, indices, and faces of all frames. We observe in Figure 5a that the geometry PSNR saturates, at relatively low bit rates, at the highest fidelity possible for a given voxel size 2^{-J}, which is just over 71 dB for $J = 10$. In Figures 5b-d we show on the *breakers* sequence that quality within 0.5dB of this limit appears to be sufficiently close to that of the original voxelization without quantization. At this quality, for *man*, *soccer*, and *breakers* sequences, the encoder in hybrid (intra-inter) mode has geometry bit rates of only 1.2, 2.7, and 2.2 Mbps, respectively. For comparison, the encoder in all-intra mode, which uses octrees for all frames, has geometry bit rates of 5.24, 6.39, and 4.88 Mbps, respectively. Thus the hybrid intra-inter mode has a geometry bit rate saving of a factor of 2-5. Further results may be found in [23].



(a) RD curves for motion compression.

(b) original (c) 62 dB (1.6 Mbps) (d) 70.5 dB (2.2 Mbps)

Fig. 5: Geometry compression.

5. CONCLUSION

We introduced polygon clouds, a new structure for representing and compressing geometry and its associated attributes, such as color. A polygon cloud is intermediate in structure between a polygonal mesh (which is more structured) and a point cloud (which is less structured). A polygon cloud has the desirable property (like a point cloud but unlike a polygonal mesh) that the topological noise and surface inconsistencies typically found in data from scanned natural objects are easy to represent. Dynamic polygon clouds also have the desirable property (like dynamic polygon meshes but unlike dynamic point clouds) that sequences are easy to compress. In fact, we introduce a method for compressing dynamic polygon cloud sequences and show that compared to static polygon clouds (and a fortiori static point clouds) dynamic polygon clouds can improve color compression by up to 2-3 dB in quality at a given bit rate, and can improve geometry compression up to a factor of 2-5 in bit rate at a given quality. We believe that dynamic polygon clouds are well suited to representing and coding the live captured content needed for emerging virtual and augmented reality systems.

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] Adrien Maglo, Guillaume Lavoué, Florent Dupont, and Céline Hudelot, "3d mesh compression: Survey, comparisons, and emerging trends," *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, pp. 44, 2015.

[4] Mukul Sati, Peter Lindstrom, and Jarek Rossignac, "ebits: Compact stream of mesh refinements for remote visualization," *Computer-Aided Design*, 2016.

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

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

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

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

[9] 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.

[10] Donald Meagher, "Geometric modeling using octree encoding," *Computer graphics and image processing*, vol. 19, no. 2, pp. 129–147, 1982.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] P. A. Chou, E. Pavez, R. L. de Queiroz, and A. Ortega, "Dynamic polygon clouds: Representation and compression for vr/ar," Technical Report MSR-TR-2016-59, Microsoft Research, Redmond, WA, USA, 2016.