Attribute compression for sparse point clouds using graph transforms

Cohen, R.A.; Tian, D.; Vetro, A.

TR2016-112 September 2016

Abstract

With the recent improvements in 3-D capture technologies for applications such as virtual reality, preserving cultural artifacts, and mobile mapping systems, new methods for compressing 3-D point cloud representations are needed to reduce the amount of bandwidth or storage consumed. For point clouds having attributes such as color associated with each point, several existing methods perform attribute compression by partitioning the point cloud into blocks and reducing redundancies among adjacent points. If, however, many blocks are sparsely populated, few or no points may be adjacent, thus limiting the compression efficiency of the system. In this paper, we present two new methods using block-based prediction and graph transforms to compress point clouds that contain sparsely-populated blocks. One method compacts the data to guarantee one DC coefficient for each graph-transformed block, and the other method uses a K-nearest-neighbor extension to generate more efficient graphs.

IEEE International Conference on Image Processing (ICIP)

ATTRIBUTE COMPRESSION FOR SPARSE POINT CLOUDS USING GRAPH TRANSFORMS

Robert A. Cohen, Dong Tian, Anthony Vetro

Mitsubishi Electric Research Laboratories
201 Broadway, Cambridge, MA 02139, USA
{cohen,tian,avetro}@merl.com

ABSTRACT

With the recent improvements in 3-D capture technologies for applications such as virtual reality, preserving cultural artifacts, and mobile mapping systems, new methods for compressing 3-D point cloud representations are needed to reduce the amount of bandwidth or storage consumed. For point clouds having attributes such as color associated with each point, several existing methods perform attribute compression by partitioning the point cloud into blocks and reducing redundancies among adjacent points. If, however, many blocks are sparsely populated, few or no points may be adjacent, thus limiting the compression efficiency of the system. In this paper, we present two new methods using block-based prediction and graph transforms to compress point clouds that contain sparsely-populated blocks. One method compacts the data to guarantee one DC coefficient for each graph-transformed block, and the other method uses a K-nearest-neighbor extension to generate more efficient graphs.

Index Terms— point cloud compression, graph transform

1. INTRODUCTION

3-D point clouds have become a practical way to represent data points in many applications, such as virtual reality, mobile mapping, scanning of historical artifacts, and 3-D printing. An advantage of using point clouds is the fact that the storage format can be decoupled from the capture method. Subsequent processing of a point cloud can then be performed in a variety of ways independently of how it was captured.

In particular, a point cloud comprises a set of coordinates or meshes, representing the point locations and some attributes attached to each point. Some typical point clouds may include native connectivity between data points, which are known as *structured* or *organized* point clouds; while some other point clouds may not have such information and are *unstructured* or *unorganized*. Note that these definitions are not standardized, as some people may refer to structured point clouds as being aligned with a rasterized grid that can correspond for example to a camera-captured image. Earlier work from the computer graphics community compressed or reduced the size of point clouds primarily by leveraging the connectivity of structured point clouds. Such approaches achieved compression by reducing the number of vertices in triangular or polygonal meshes by, for example, fitting surfaces or splines to the meshes. Surveys of many of these methods can be found in [1], [2], and [3].

Considering the large size and scale of today's point clouds, block-based and hierarchical octree-based methods have been utilized to perform compression on a block-by-block basis, both for improving coding efficiency and reducing hardware or software complexity. Peng [4] showed an octree representation to code structured point clouds, which has been extended for coding unstructured point clouds in real time in a recent work in [5]. That work was implemented using the open-source Point Cloud Library (PCL) [6]. A tutorial on using PCL to compress point clouds can be found in [7].

For compressing 2-D images and videos, significant progress has been made over the past several decades, resulting in widely used standards such as JPEG [8], H.264/AVC [9] and HEVC [10]. The same principles of block-based and hierarchical processing used in image and video coding systems can be extended to work for point cloud data. By mapping data from the point clouds into 2-D arrangements, existing image and video coders can be employed for representation and compression. For example, Xu [11] proposed to compress point clouds in a teleoperation environment based on H.264/AVC. Mekuria [12] extended the PCL-based approach in [5] by using a JPEG coder on blocks of attributes, which achieved substantial improvements in coding efficiency. Recently, Zhang [13] developed a graph transform for coding blocks of point cloud attributes, but it required the point cloud to be captured or arranged onto a regular grid. In [14], point clouds were downsampled to a uniform grid, which in turn was partitioned into blocks so that the graph transform could be directly applied. The latter work also included 3-D intra-block prediction.

In this paper, we develop two ways to improve coding efficiency when graph transforms are used for lossy compression on blocks partitioned from a large or sparse point cloud. In Section 2, we give an overview on how point clouds can be partitioned into blocks and how graph transforms have been used to compress attributes contained within the blocks. Section 3 shows how data in a block can be compacted for more efficient coding. In Section 4, we show that by expanding the concept of adjacency when allowing the graph to include K nearest neighbors, the number of graphs used to code a block can be reduced. Experimental results are shown in Section 5, and a summary and conclusions are given in 6.

2. POINT CLOUD PARTITIONING AND ATTRIBUTE COMPRESSION USING GRAPH TRANSFORMS

We can partition a 3-D point cloud into blocks by either capturing the data in way that outputs voxels aligned to a grid, or we can divide the point cloud by a uniform grid having a known resolution. In [15], a sparse voxelization approach is used to capture data and arrange it on a 3-D grid where each direction has dimensions 2^j, and j indexes a voxel hierarchy. With the data arranged on a grid, the method described in [13] applies a graph transform to the attributes in each block. For point clouds that are not necessarily arranged on a grid, the work in [14] resamples the points to lie on a uniform grid, and then the grid is partitioned into 3-D blocks of size $k \times k \times k$. A modified shape-adaptive transform is then used to compress the attributes within each block. For both these methods, and for this paper, it is assumed that the (x, y, z) point positions are coded separately before coding the attributes, so the focus of these works is on compressing the attributes associated with the points.

The resampling method used in [14] decomposes a point cloud to an octree representation having a minimum voxel resolution r, where r represents the minimum length of an edge of the voxel corresponding to a leaf node of the octree. If the sensor or capture resolution of the point locations in the point cloud is less than r, then a leaf node may contain more than one point. The attribute associated with the leaf node is set to the average over all points in the node. The geometric center of this leaf voxel becomes the (x, y, z) location associated with this averaged attribute. By using the geometric center, this ensures that the resampled point cloud lies on a uniform grid of resolution r, which in turn can easily be partitioned into blocks. If, however, we want to preserve all the attributes and not average them, then the octree resolution r would need to be set to the sensor resolution to guarantee that each octree leaf node contained only one point from the input point cloud. Partitioning these finely-sampled positions into $k \times k \times k$ blocks for small k may result in many blocks that contain very few samples. For larger k, the blocks may be sparsely populated, or they may contain many non-adjacent groups of points.

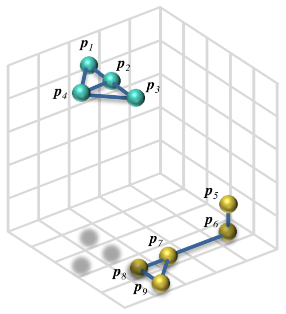
The graph transform, as described in [13], computes an adjacency matrix $\boldsymbol{A}$ and then populates the matrix with nonzero weights to indicate which points are considered to be adjacent. A graph Laplacian matrix $\boldsymbol{Q}$ is computed based on the weights, and then the eigenvector matrix of $\boldsymbol{Q}$ is applied as a transform on the attribute values. An example from [14] illustrating the construction of an adjacency matrix is shown in Fig. 1. Because the structure of the graph transform is based upon the adjacency matrix, the transform associates the equivalent of one DC coefficient and corresponding AC coefficients to every isolated point and isolated groups of adjacent points. For a block having many isolated points, this structure produces many DC coefficients; much more than if a more connected graph transform were applied. As the block size increases, namely as the block partitioning resolution of the point cloud increases, coding efficiency will be severely impacted. We will show evidence of these effects in Section 5. To ameliorate these effects, we introduce two methods: compacting the block prior to applying the graph transform, and expanding the definition of "adjacent" to include K nearest neighbors when constructing the graph.

3. COMPACTING BLOCKS

A diagram of the compacting process is shown in Fig. 2. This process has similarities to the shifting process used by the shape-adaptive DCT (SA-DCT) [16]. Unlike [16], however, we apply it to the input points before performing the graph transform. Also, we do not need to compute and signal the boundary shape of the attributes, because our "shape" is defined by the presence of a point occupying an available element of the block. These points could be considered as being the foreground, and elements of the $k \times k \times k$ block that do not contain points can be considered the background. The shifting process shifts out empty elements toward one border in each dimension. The end result of this process is that all points are compacted toward one corner of the block. This guarantees that all points are part of one graph when the graph transform is applied. Therefore, only one DC coefficient is produced, along with corresponding AC coefficients. There is a trade-off inherent in this process: We are gaining coding efficiency by changing the positions of the points so they can be compacted with one transform, but we are potentially losing the spatial relationship among the points when processing the attributes. Note, however, that this repositioning is only done for applying a transform to the attribute values. The decoder reverses this compaction process to reconstruct the point cloud, ensuring that each decoded attribute is assigned to its corresponding spatial position. The compacting process is repeated over all blocks, until the entire point cloud is compressed. Blocks not containing points can be ignored, as no attributes need to be compressed for them.

4. GRAPH TRANSFORM WITH K NEAREST NEIGHBORS

As described in Section 3, the compacting method relocates the points in the block, which may make it difficult to leverage the correlation among neighboring points before they are



$$\boldsymbol{A} = \begin{pmatrix} 0 & w_{12} & 0 & w_{14} & 0 & 0 & 0 & 0 & 0 \\ w_{12} & 0 & w_{23} & w_{24} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{23} & 0 & w_{34} & 0 & 0 & 0 & 0 & 0 \\ w_{14} & w_{24} & w_{34} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{56} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{56} & 0 & w_{67} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{67} & 0 & w_{78} & w_{79} \\ 0 & 0 & 0 & 0 & 0 & 0 & w_{78} & 0 & w_{89} \\ 0 & 0 & 0 & 0 & 0 & 0 & w_{79} & w_{89} & 0 \end{pmatrix}$$

Fig. 1: Example of graph transform applied to block containing two separate groups of points, and the corresponding adjacency graph, from [14]

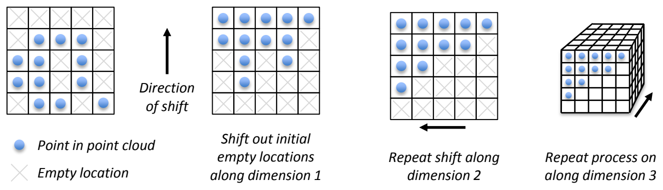

Direction of shift
Point in point cloud
Empty location
Shift out initial empty locations along dimension 1
Repeat shift along dimension 2
Repeat process on along dimension 3

Fig. 2: Compacting blocks prior to applying the graph transform

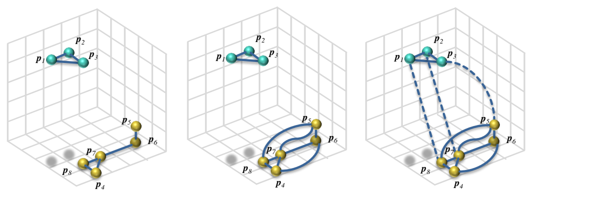


Fig. 3: Illustration of process going from separate sub-graphs to one graph by using K-NN with $K = 3$

moved. As a consequence, a graph constructed on the compacted block is not guaranteed to improve coding efficiency, especially if the original block could be represented with one graph.

The graph transform in [13] is limited in that assumes that the points are aligned to grid positions, and only the points that are one unit apart in any dimension are identified as being connected when constructing the graph. This approach does not permit multiple attributes to be co-located on one grid point, so resampling or attribute averaging of those points would be needed. Sampling the grid at a finer resolution would avoid this problem, but it could lead to many disjoint sub-graphs in each block, which can impact coding efficiency.

In this section, we construct graph edges to include more distant points, located on quantized or fractional point positions. Instead of limiting neighbors to being one unit apart, we connect each point to its K nearest neighbors and then prune any duplicated edges. By allowing connections to more distant points, this K-nearest-neighbors method (K-NN) is able to incorporate more points into each graph, thus reducing the total number of disjoint graphs in a sparse block.

Fig. 3 shows the process of constructing the K-NN graph with $K = 3$, where the K-NN approach will make new connections (P_1, P_8), (P_2, P_7) and (P_3, P_5), as compared to the graph constructed using [13]. However, unlike the compacting approach of Section 3, the K-NN graph is not guaranteed to encompass all points of a block in one sub-graph. Additional processes could be developed and subsequently applied to determine how to connect multiple disjointed sub-graphs. It is worth noting that in the K-NN graph after pruning, there may exist some points which are connected to more than K points. For example, P_7 in Fig. 3 has 5 associated edges because P_7 is a nearest neighbor point for some extra points, P_2 and P_3, in addition to its own nearest neighbors.

Finally, the weight of the graph edge is a function of distance, e.g. $e^{-\frac{d}{2\sigma^2}}$, where d is a measured distance between the connected points and σ denotes the variation in statistics of the points. In the next section, we will discuss the results of experiments using several methods, for different block sizes, partition resolutions, and values of K.

5. EXPERIMENTAL RESULTS

The input point cloud used for experiments is the unconnected *Statue_Klimt_PointCloud.ply* point cloud described in [17]. For this paper we use the first 3000 points to keep the simulation times practical. Each point of the input point cloud comprises a floating-point (x, y, z) location and a corresponding RGB color attribute. We convert the RGB color attributes to YCbCr [18] luminance and chrominance values and use the 8-bit luminance value Y as the attribute to be coded. We apply the octree decomposition described in [14] with an octree resolution of 0.0081, which generates octree leaf nodes having only one point per node. We used the center position of each node as the (x, y, z) coordinate for the corresponding attribute. This process allows us to align the data into rows and columns on a fine grid without downsampling. The point cloud is next divided into blocks based on the partition resolution pr. For these experiments, we use pr = 0.15, 0.3, and 0.5. For all cases we incorporate the

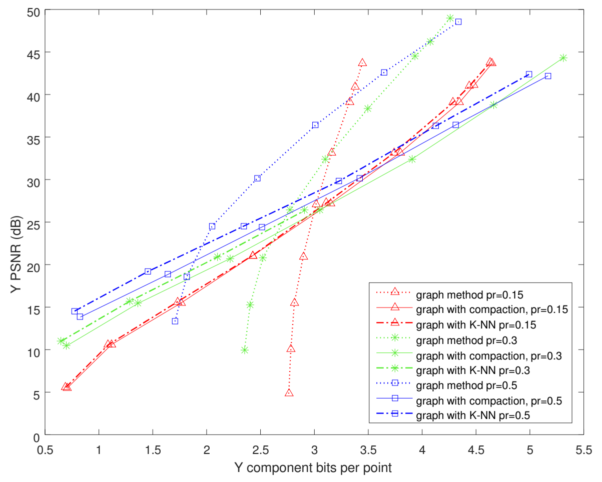


Fig. 4: Coding performance of the three graph transform methods

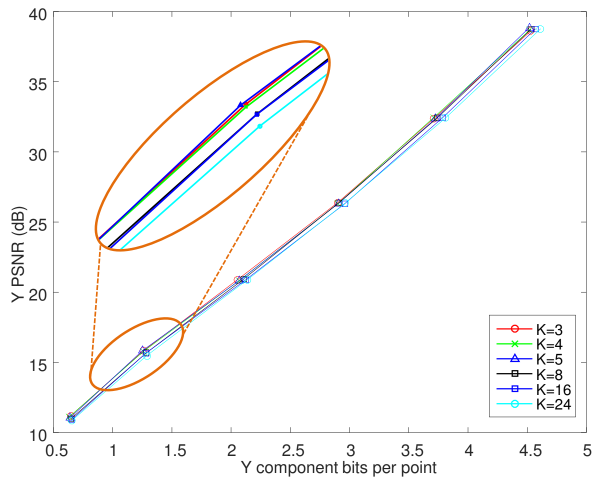


Fig. 5: Performance of K-nearest-neighbor graph transform for different values of K with $pr = 0.5$

intra-block prediction of [14]. With an octree resolution of 0.0081, the maximum block sizes associated with these values of pr are approximately 18, 37, and 61, respectively. As described [13] and [14], an arithmetic coder using a Laplacian model is used to entropy code the coefficients. Plots showing luminance PSNR vs. the rate in bits per point are shown in Fig. 4. The three methods used here are the unmodified graph transform from earlier works, where elements were adjacent if they were within one unit in any dimension; the method of Section 3 which compacts the points to one corner of the block before applying the transform; and the K-NN graph transform method of Section 4 with $K = 8$.

In Fig. 4 we can see that at the lower rates corresponding to coarse levels of quantization, the performance of both the compacting and K-NN methods exceeds that of the unmodified graph-transform. As the rate increases, the performance of the graph-transform method eventually exceeds the performance of the other two methods. The bit-rate at which this crossover point occurs increases as the partition resolution or block size decreases. The primary reason for the improvement in performance of the new methods is that when coding a sparse block where most of the points are not adjacent, the unmodified graph-transform method can produce dozens of DC coefficients that need to be quantized and coded. The compacting method generates only one DC coefficient per block, due to its design. For these experiments, the K-NN method with $K = 8$ typically produces only one or two DC coefficients per block. With these modified transforms, a coarse quantizer typically zeroes out the AC coefficients but preserves the DC coefficient. With the unmodified transform, the large set of DC coefficients is coarsely quantized, resulting in an increased distortion. As the rate increases, eventually the fidelity of the reconstructed DC coefficients is sufficient to yield an improved coding performance. As the block size decreases, the total number of DC coefficients for the whole point cloud increases for the two new methods, thus reducing their overall coding performance. As the block size decreases for the K-NN transform, the number of disjoint sub-graphs in each block will decrease, which in turn reduces the number of DC coefficients per block. This behavior can be observed in Fig. 4, in that with decreasing block size, the performance of the K-NN transform approaches that of the compacting method, which always has one DC coefficient.

Compression performance curves for varying K are shown in Fig. 5 for partition resolution $pr = 0.5$. Generally, at the lower rates where the new methods work well, using K=3, 4, or 5 neighbors works best. At higher rates there is not a significant difference in performance among these values of K. For the cases tested here, increasing K above 8 decreased performance by about 0.5 dB.

6. SUMMARY AND CONCLUSIONS

In this paper, we extended some of the concepts used to code images and video to compress attributes from unstructured point clouds that have been partitioned into blocks without downsampling, thus preserving all the attributes. By compacting the blocks or modifying the adjacency definition of the graph transform to include up to K nearest neighbors without a distance limitation, we reduced the number of DC coefficients as compared to the unmodified graph transform in which almost every point may become a DC coefficient due to the sparseness of the block. Experimental results were shown for these methods over different partition resolutions and numbers of K nearest neighbors. In addition to a more detailed study of performance over a wider variety of parameters and point clouds, future work also will include more advanced ways of connecting sub-graphs within a block, e.g. by combining distance restrictions and numbers of nearest neighbors; or by considering other structural characteristics of the points in each block to control compacting and graph generation.

7. REFERENCES

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

[2] Pierre Alliez and Craig Gotsman, "Recent advances in compression of 3D meshes," in *In Advances in Multiresolution for Geometric Modelling*. 2003, pp. 3–26, Springer-Verlag.

[3] Adrien Maglo, Guillaume Lavoué, Florent Dupont, and Céline Hudelot, "3D mesh compression: Survey, comparisons, and emerging trends," *ACM Comput. Surv.*, vol. 47, no. 3, pp. 44:1–44:41, Feb. 2015.

[4] Jingliang Peng and C.-C. Jay Kuo, "Geometry-guided progressive lossless 3D mesh coding with octree (OT) decomposition," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 609–616, July 2005.

[5] J. Kammerl, N. Blodow, R.B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach, "Real-time compression of point cloud streams," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, May 2012, pp. 778–785.

[6] Radu Bogdan Rusu and Steve Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, CN, May 2011.

[7] J. Kammerl, "Point cloud compression," `http://pointclouds.org/documentation/tutorials/compression.php#octree-compression`, [Online; accessed 2015-11-09].

[8] ITU-T and ISO/IEC, *Digital compression and coding of continuous-tone still images*, ITU-T Rec. T.81 and ISO/IEC 10918-1 (JPEG), Sept. 1992.

[9] ITU-T and ISO/IEC JTC1, *Advanced Video Coding for Generic Audio-Visual Services*, ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC), May 2003 (and subsequent editions).

[10] G. J. Sullivan, J. Ohm, Woo-Jin Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, 2012.

[11] Xiao Xu, Burak Cizmeci, and Eckehard Steinbach, "Point-cloud-based model-mediated teleoperation," in *Proc. of IEEE Int. Symposium on Haptic Audio-Visual Environments and Games (HAVE)*, Istanbul, Turkey, Oct. 2013.

[12] ISO/IEC JTC1/SC29/WG11, "Point cloud encoder, decoder and comparative results," document MPEG2015/m35910, ISO/IEC JTC1/SC29/WG11 *Coding of Moving Pictures and Audio*, Geneva, Switzerland, Feb. 2015.

[13] Cha Zhang, Dinei Florencio, and Charles Loop, "Point cloud attribute compression with graph transform," in *2014 IEEE International Conference on Image Processing (ICIP)*, Oct. 2014.

[14] Robert A. Cohen, Dong Tian, and Anthony Vetro, "Point cloud attribute compression using 3-D intra prediction and shape-adaptive transforms," in *Proc. 2016 Data Compression Conference, DCC 2016*, Snowbird, UT, USA, March 29-April 1 2016.

[15] Charles T. Loop, Cha Zhang, and Zhengyou Zhang, "Real-time high-resolution sparse voxelization with application to image-based modeling," in *Proceedings of the 5th High-Performance Graphics 2013 Conference HPG '13*, Anaheim, California, USA, July 2013, pp. 73–80.

[16] T. Sikora and B. Makai, "Shape-adaptive DCT for generic coding of video," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, no. 1, pp. 59–62, Feb. 1995.

[17] Veronica Scurtu, Christian Tulvan, Adrian Gabrielli, and Marius Preda, "Reconstruction platform and datasets for 3D point-cloud compression," document MPEG2015/m36523, ISO/IEC JTC1/SC29/WG11 *Coding of Moving Pictures and Audio*, Warsaw, Poland, June 2015.

[18] ITU-R Recommendation BT.601-7, "Studio encoding parameters of digital television for standard 4:3 and wide screen 16:9 aspect ratios," March 2011.