G-ARRAYS: GEOMETRIC ARRAYS FOR EFFICIENT POINT CLOUD PROCESSING

Hoda Roodaki, Masoud Dehyadegari

K. N. Toosi University of Technology
{hroodaki,dehyadegari}@kntu.ac.ir

Mahdi Nazm Bojnordi

School of Computing, University of Utah
bojnordi@cs.utah.edu

ABSTRACT

With the increasing demand for 3D modeling by the emerging immersive applications, the 3D point cloud has become an essential representation format for processing 3D images and video. Because of the inherent sparsity in 3D data and the significant memory requirements for representing points, point cloud processing is a challenging task. In this paper, we propose a novel data structure for representing point clouds with a reduced memory requirement and a faster lookup than the state-of-the-art formats. The proposed format is examined for temporal encoding in geometric point cloud compression. Our simulation results show that the proposed temporal prediction enhances the compression rate and quality by 13-33% as compared to MPEG G-PCC. Moreover, the proposed data structure provides 16-54× faster point lookup operations and more than 1.4× reduction in memory consumption compared to the octree structure used in the MPEG G-PCC.

Index Terms— Point cloud processing, spatial/temporal coding, memory management.

1. INTRODUCTION

3D point clouds are a collection of individual 3D points where, each point includes 3D coordinates along with several attributes such as color and reflection [1] [2] [3]. Point clouds are a crucial approach to represent 3D objects in autonomous driving [4] [5] [6] [7], virtual and augmented realities [8], medical imaging, indoor navigation [9], and real-time 3D telepresence [10]. Due to the significant memory requirements of point clouds, compression techniques are becoming inevitable. Efficient compression of point cloud data with minimal negative impact on the quality of video is crucial [11][1]. Without compression, a 3.6 Gbps bandwidth is necessary to transfer 30 frames per second while each frame comprises 1 million points [11]. Some efforts have been made to compress point clouds for various communication networks. MPEG has proposed Video-based (V-PCC) and Geometry-based (G-PCC) approaches for point cloud compression [11].

Advanced compression methods for point clouds are based on an octree structure [12] [13] to remove the spatial and temporal redundancy within a point cloud [14]. These methods try to remove the temporal redundancies of point cloud streams by comparing the octree differences of consecutive point cloud frames. For the temporal encoding of a frame (a.k.a., predicted frame), instead of directly encoding the raw pixel values, the encoder finds similar points to the points of the predicted frame from a previously encoded frame (a.k.a., reference frame). The proposed method relies on estimating motion between blocks of the consecutive frames. In this process, given a specific block A in the predicted frame and its matching block B in the reference frame, a motion vector (MV) is defined as a vector connecting the top-left coordinates of A and B. Then, the extracted motion vector and a residual block are encoded and sent to the decoder. The residual block is defined as the difference between A and B.

The geometry of points in point clouds, unlike the computer-generated or natural 2D videos, has limited spatio-temporal locality [15]. Figure 1 shows two consecutive frames captured at different time instances for the 3D model of Soldier [16]. Each frame has a different number of points. Moreover, there is no explicit correspondence among the points of the two frames. This makes extracting the temporal redundancy between successive frames a challenging task.

To estimate motion among 3D blocks, we need to store the points in a searchable data structure. The data structure may be a sparse matrix because the point cloud doesn't necessarily include all possible points in the 3D space. Storing the non-existing points would lead to polluting the memory. Even in its sparse representation form, a typical point cloud comprises millions of points, which imposes a significant pressure on the memory capacity and bandwidth. The state-of-the-art methods propose to use an octree structure for this purpose [17]. To represent a 3D point cloud using the octree structure, the 3D space is divided into eight octants. Each octant can be further divided to smaller octants if it has more than one point. The block division for forming an octree is repeated based on a user-defined parameter, namely depth [18]. The deeper the octree, the finer-grained accesses to the points are enabled. The most common implementation of the octree employs a tree structure with pointers to represent hierarchical connections among the octants, i.e., each node of the octree has eight pointers, one for each of its children, and a reference to the associated data. The large number of pointers used in the octree results in a significant memory consumption and requires numerous memory indirections for each point access that makes the point cloud processing slow. As an efficient alternative to octrees, we propose a novel fine-grained searchable storage format using decoupled geometric arrays (G-arrays) that enable a faster lookup than the state-of-the-art octrees. The use of G-arrays for temporal encoding in geometric point cloud compression can save memory consumption by an average of 1.4×.

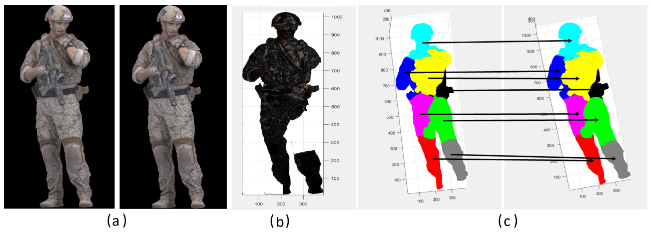


Fig. 1. (a) Two consecutive frames of the point cloud sequence, Soldier, captured at different time instances (b) the Residual frame, and (c) the obtained motion vector.

2. RELATED WORK

The G-PCC reference encoder in [11] [19] encodes the data directly in the 3D space using an octree to describe the location of points for spatial prediction. Geometry encoding includes *coordinate transformation*, *voxelize*, *geometry analysis*, and arithmetic coding. Kammerl et al. [20] remove the temporal redundancies of input point cloud streams by comparing the octrees of consecutive frames. Mekuria et al. [10] exploit the octree structure for spatial and temporal coding to compress real-time 3D tele-immersive video. Thanou et al. [15] present an approach for motion estimation and compensation for geometry and color information to provide a compression framework for 3D point clouds.

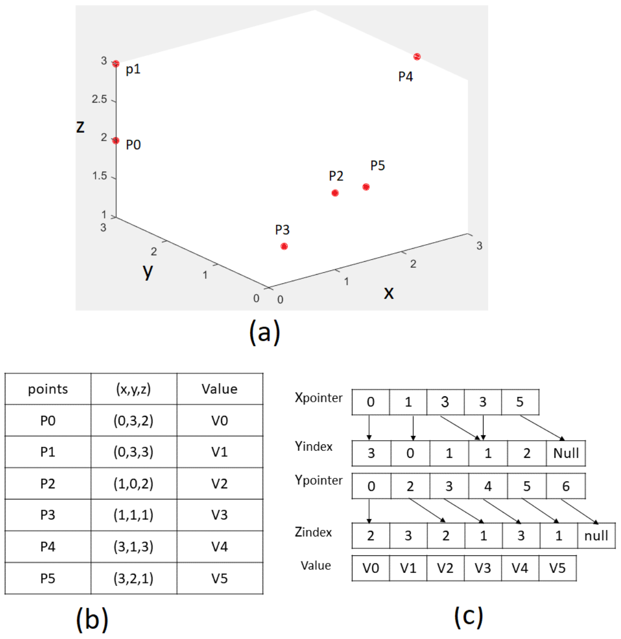
3. PROPOSED METHOD

One of the main difficulties in G-PCC coding is representing the intermediate point cloud data for motion estimation. First, we propose an efficient data representation for 3D point cloud based on geometric arrays. Then, we examine the use of G-arrays in a cluster-based temporal prediction mechanism.

3.1. Geometric Arrays

To process point cloud for motion estimation, the X, Y, and Z coordinates of the points should be traversed to find the existing points. In a temporal prediction process, to compare each point of the predicted frame with the points of the reference frame, the points of a cubical region in the reference frame should be traversed. Figure 2 shows an example of the G-arrays data layout for six points (i.e., *P0* to *P5*) with different coordinates. We have five arrays in this format.

- A $Value$ array stores the attribute values of the existing points in the point cloud.
- A Z_{index} vector shows the Z coordinates of the corresponding points in the $Value$ array.



points	(x,y,z)	Value
P0	(0,3,2)	V0
P1	(0,3,3)	V1
P2	(1,0,2)	V2
P3	(1,1,1)	V3
P4	(3,1,3)	V4
P5	(3,2,1)	V5

Fig. 2. Presentation of (a) six points in a 3D space, (b) a list of the points, and (c) the points in the proposed G-arrays.

- A Y_{index} vector shows the Y coordinates of the points in the $Value$ array.
- A $Y_{pointer}$ vector stores the cumulative number of existing points with the Y coordinate equal to $Y_{index}[i]$. It is defined by the following recursive relation.
  - $Y_{pointr}[0] = 0$
  - $Y_{pointer}[i+1] = Y_{pointer}[i]$ + number of existing points with their Y coordinates equal to $y_{index}[i]$
- An $X_{pointer}$ vector stores the cumulative number of existing points with the X coordinate equal to the index of $X_{pointer}$ vector. It is defined by the recursive relation below.
  - $X_{pointer}[0] = 0$
  - $X_{pointer}[i+1] = X_{pointer}[i]$ + number of existing points with X coordinate equal to i and non equal Y values

3.2. Point Lookup

To access a specific point $P = (p, q, r)$ in G-arrays, we should first determine if this point exists or not. For this purpose, it is enough to refer to index p of $X_{pointer}$ array. If $X_{pointer}[p+1] - X_{pointer}[p]$ is non-zero, there exist at least one point with the X coordinate equal to P. Then, the pointers of $X_{pointer}[p]$ and $X_{pointer}[p+1]$ show the candidate locations for finding the corresponding Y coordinates in the y_{index} array. Searching the cells with indices between these two pointers in the Y_{index} array, the point with X and Y coordinates equal to p and q could be found. Then, the index of the point found in $Y_{pointer}$ array (i.e., k) is used to find the corresponding Z coordinate. The $Y_{pointer}[k+1] - Y_{pointer}[k]$ shows the number of points with X and Y coordinates equal to p and q respectively and the pointers of $Y_{pointer}[k]$ and $Y_{pointer}[k+1]$ in $Y_{pointer}$ array show the candidate locations for finding the corresponding Z coordinate in the Z_{index} array.

3.3. Temporal Prediction with G-Arrays

Figure 3 shows the proposed architecture for 3D point cloud-temporal compression. First, we consider the first frame of any input sequence as the reference for the second frame. We employ a spatial coding for the first frame based on the prior work on MPEG G-VCC [17]. Then, we consider the second path for all the other frames as predicted frames, which includes several steps for clustering points, estimating cluster motion, and generating point residual.

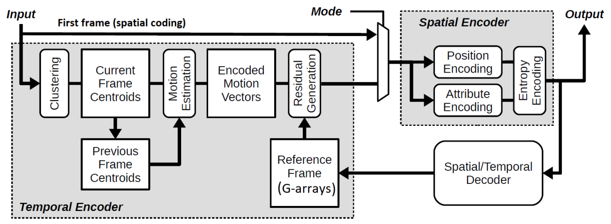


Fig. 3. Our proposed point cloud spatio-temporal encoding based on G-arrays.

3.3.1. Clustering Points

The first step in the proposed temporal prediction technique is *clustering* the points of the reference and predicted frames. The clustering outcome is used to find the motion vectors. The idea is that by clustering the consecutive frames, parts of the frame with high similarity are most likely placed in the corresponding clusters of reference and predicted frames. In the motion estimation process, for each point of the predicted frame, we want to look for the most similar point in the reference frame. Hence, the corresponding cluster of each point in the reference frame is enough to find the most similar point.
The k-means algorithm is used for clustering of the reference and predicted frames. The right number of clusters in k-means algorithm depends on a trade-off between the number of distance computations and the quality of clustering. We consider eight initial center points at the corners of each frame based on the minimum and maximum values of the X, Y and Z coordinates.

3.3.2. Estimating Cluster Motion

Motion estimation is used to describe the movement of textures or objects in consecutive frames. When the best matching block for a predicted block is found in the reference frame, the corresponding movement between the reference and predicted blocks is represented by motion vector.
In the proposed approach, we employ a coarse-grained motion vector estimation algorithm. This algorithm uses the centroids of the computed clusters in the previous step to estimate the motion vectors as shown in Figure 4. Each cluster contains points with similar coordinates. Hence, the amount of changes in the centroids of the corresponding clusters from the reference and predicted frames could be considered as a good estimation of the amount of movement between the points. Accordingly, the motion vectors are computed using the centroids shown in Figure 4. Then, the motion vectors are coded as part of the output bitstream.

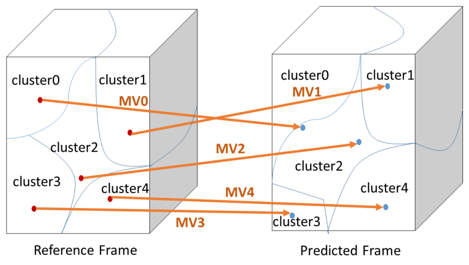


Fig. 4. Finding motion vectors based on the clustering of points in the reference and predicted frames.

3.3.3. Generating Point Residual

After computing motion vectors, a reconstructed frame using the predicted frame and the estimated motion vectors is generated, the points of each cluster are displaced according to its corresponding motion vector.
Finally, we should find the reconstructed error frame, often referred to as the residual frame. The residual frame contains the information to correct the error of motion estimation process. For this purpose, the difference between the attributes of the corresponding points in the reference and reconstructed frames should be computed. But, the main issue is how to find the corresponding point to each point of the reconstructed block in the reference frame. Our proposed scheme suggests to represent the reference frame in G-arrays to search its points more quickly. A matching criterion such as Mean Squared Error (MSE) or Sum of Absolute Differences (SAD) can be used to find the best MV per block. SAD is preferable for VLSI implementation because of its simple computational steps. Therefore, we use SAD for choosing the best MV for a given block. After searching, the smallest SAD candidate is chosen as the best matching reference for each point of the reconstructed frame. Then, the residual frame is created by calculating the difference between the RGBs of the reconstructed and the reference frame points.

4. EVALUATIONS

In this section, we assess the performance of the proposed compression scheme using G-arrays and point clustering. We use four different sequences that capture human bodies in motion, i.e., Soldier, Longdress, Loot, and Redandblack [16]. We study the benefits of temporal prediction in the compression of geometry in 3D point cloud sequences. In our proposed approach, the frames of the sequences are coded sequentially. Each frame is coded in a way that its previous frame is considered as a reference frame. For the baseline method, all of the frames are coded using their best set of parameters in the MPEG G-PCC open-source library [17]. We measure performance in terms of the output quality, compression ratio, memory requirement, and lookup speedup. There are several objective quality metrics available in the literature to measure the quality of encoded point clouds. One is a point-to-point metric that measures the quality using the point-based distances. For each point in the decoded frame, the nearest neighbor is obtained in the original point cloud frame and a distance-based mean squared error (MSE) is computed over all pairs of points. But, since, the point cloud points represent the surface of objects in a visual scene, a point-to-plane metric is often used to better evaluate the output quality. The point-to-plane metric approximates the underlying surface at each point as a plane. This metric results in smaller errors for the points that are closer to the surface [21]. We study various quantization parameters (QP = 11, 10, and 9) for our simulations. Where the quantization parameter determines the number of bits representing each component of the points. The PSNR versus compression ratio curves are shown in Figure 5 to compare the performance of our method with that of the baseline. The results show that the proposed method improves the compression ratio for various output qualities over the baseline MPEG G-PCC.

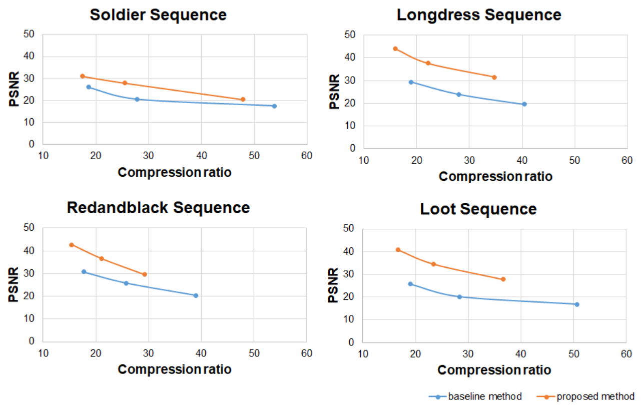


Fig. 5. PSNR versus the compression ratio for various test sequences for baseline method and our proposed method.

To further evaluate the performance of the proposed temporal encoding, we consider a representative metric based on both quality and the compression ratio. The goal is to simultaneously increase the compression ratio and quality. The new metric is defined as the product of the compression ratio and PSNR. Figure 6 shows the comparison of the baseline and our proposed method based on this metric. The results indicate that the proposed method improves the compression ratio and quality by averages of 13-30% over the MPEG G-PCC.

We also evaluate the performance of the G-arrays as compared to octrees in terms of memory consumption and point lookup speed. Table 1 shows the memory requirements of our method compared to the baseline method. Our method uses the G-arrays to store the points and search for the best matching ones. But, the baseline method uses the octree structure. We observe more than 1.4× memory savings for the G-arrays over the octrees. Moreover, the proposed G-arrays enable 16-54× better point lookup speed as compared to the state-of-the-art octrees. Our simulation results indicate that G-arrays can replace octrees in point cloud processing to enable significantly faster point lookups and lesser memory consumption. Even in the existing systems that are not designed to use the G-arrays as an alternative to octrees, we can use the proposed G-arrays in the post-processing stages. Before or after encoding/decoding a point cloud sequence, further post-processing operations may be applied to the 3D point cloud, such as reshaping objects, merging clouds, and adding new points to obtain a smoother point cloud.

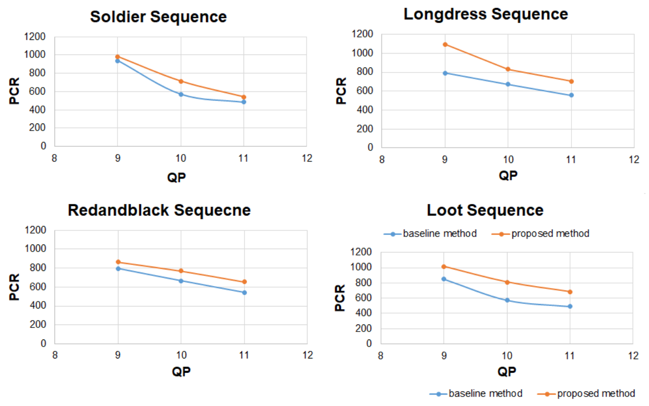


Fig. 6. The product of PSNR and compression ratio (PCR) for various quantization parameters (QP) for baseline method and our proposed method.

Table 1. Memory savings and performance gains of the proposed G-arrays over the state-of-the-art octree.

Sequence name	Memory saving over octree	Lookup speedup over octree
Soldier	1.44×	16.59×
Longdress	1.43×	15.97×
Redandblack	1.42×	23.58×
Loot	1.40×	54.78×

5. CONCLUSIONS

In this paper, we presented a novel data structure for processing point clouds, called G-arrays, that relies on five arrays. A cluster-based temporal prediction approach was examined using the proposed G-arrays. Our simulation results on a set of point cloud sequences showed that the proposed format provides faster point lookup operations and less memory consumption compared to the state-of-the-art octree used by the MPEG G-PCC. Moreover, the proposed temporal prediction approach improves the compression ratio and quality of the geometric point clouds.

6. REFERENCES

[1] M. Hosseini and C. Timmerer, “Dynamic adaptive point cloud streaming,” in *Proceedings of the 23rd Packet Video Workshop*, 2018, pp. 25–30.

[2] S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P Cesar, P. A Chou, R. A Cohen, M. Krivokuća, S. Lasserre, Z. Li, et al., “Emerging mpeg standards for point cloud compression,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 1, pp. 133–148, 2018.

[3] F. Qian, B. Han, J. Pair, and V. Gopalakrishnan, “Toward practical volumetric video streaming on commodity smartphones,” in *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, 2019, pp. 135–140.

[4] A. H Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, “Pointpillars: Fast encoders for object detection from point clouds,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12697–12705.

[5] M. Liang, B. Yang, S. Wang, and R. Urtasun, “Deep continuous fusion for multi-sensor 3d object detection,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 641–656.

[6] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, “Frustum pointnets for 3d object detection from rgb-d data,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 918–927.

[7] S. Wang, S. Suo, W.-C.hiu Ma, A. Pokrovsky, and R. Urtasun, “Deep parametric continuous convolutional neural networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2589–2597.

[8] S. Zhao, H. Zhang, S. Bhuyan, C. Subhra Mishra, Z. Ying, M.T. Kandemir, A. Sivasubramaniam, and C.R. Das, “Déja view: Spatio-temporal compute reuse for energy-efficient 360 vr video streaming,” 2020.

[9] Y. Zhu, R. Mottaghi, E. Kolve, J.J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, “Target-driven visual navigation in indoor scenes using deep reinforcement learning,” in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3357–3364.

[10] R. Mekuria, k. Blom, and Cesar P., “Design, implementation, and evaluation of a point cloud codec for tele-immersive video,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 4, pp. 828–842, 2016.

[11] D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, and A. Tabatabai, “An overview of ongoing point cloud compression standardization activities: video-based (v-pcc) and geometry-based (g-pcc),” *APSIPA Transactions on Signal and Information Processing*, vol. 9, 2020.

[12] D. Meagher, “Geometric modeling using octree encoding,” *Computer graphics and image processing*, vol. 19, no. 2, pp. 129–147, 1982.

[13] J. Elseberg, D. Borrmann, and A. Nüchter, “One billion points in the cloud–an octree for efficient processing of 3d laser scans,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 76, pp. 76–88, 2013.

[14] R. Schnabel and R. Klein, “Octree-based point-cloud compression,” *IGGRAPH 2006*, vol. 6, pp. 111–120, 2006.

[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, 2016.

[16] Krivokuca M., Philip A.C., , and Savill P., “8i voxelized surface light field (8ivslf) dataset,” 2018.

[17] “Mpeg.accessed: october 2020. [online]. available: https://github.com/mpeggroup/mpeg-pcc-tmc13,” .

[18] Soohee H., “Towards efficient implementation of an octree for a large 3d point cloud,” *Sensors*, vol. 18, no. 12, pp. 4398, 2018.

[19] K. Mammou, P. A. Chou, D. Flynn, M. Krivokuća, and O. Nakagami, “G-pcc codec description v4,” *ISO/IEC JTC1/SC29/WG11 N18189*, 2019.

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

[21] Tian D., Ochimizu H., Feng C., Cohen R., and Vetro A., “Geometric distortion metrics for point cloud compression,” 2017.