Point Cloud Attribute Compression using 3-D Intra Prediction and Shape-Adaptive Transforms

Robert A. Cohen, Dong Tian, and Anthony Vetro

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

Abstract

With the increased proliferation of applications using 3-D capture technologies for applications such as virtual reality, mobile mapping, scanning of historical artifacts, and 3-D printing, representing these kinds of data as 3-D point clouds has become a popular method for storing and conveying the data independently of how it was captured. A point cloud consists of a set of coordinates indicating the location of each point, along with one or more attributes such as color associated with each point. Because the size of point cloud data can be quite large, compression is needed to efficiently store or transmit this data. This paper, motivated by techniques currently being used for image and video coding, proposes methods using 3-D block-based prediction and transform coding to compress point cloud attributes. Experimental results using a modified shape-adaptive DCT tailored for use in 3-D point clouds and a benchmark using 3-D graph transforms are shown.

1. Introduction

With the recent advancements and reductions in cost of 3-D capture technologies, there has been an increasingly wide proliferation of 3-D applications such as virtual reality, mobile mapping, scanning of historical artifacts, and 3-D printing. These applications use different kinds of sensors to capture the world in three dimensions, producing massive amounts of data. Representing these kinds of data as 3-D point clouds has become a practical method for storing and conveying the data independently of how it was captured.

A point cloud consists of a set of coordinates or meshes indicating the location of each point, along with one or more attributes such as color associated with each point. Point clouds that include connectivity information among vertices are known as *structured* or *organized* point clouds. Point clouds that contain point locations without connectivity information are *unstructured* or *unorganized*. Much of the earlier work in reducing the size of point clouds, primarily structured, has come from the computer graphics community. Many of those approaches achieved compression reducing the number of vertices in triangular or polygonal meshes, for example by fitting surfaces or splines to the meshes. Surveys of many of these methods can be found in [1], [2], and [3].

Block-based and hierarchical octree-based approaches have also been used to compress point clouds. For example, octree representations were used to code structured point clouds (meshes) in [4], and more recent work on coding unstructured point clouds in real time is described in [5]. The work of latter paper was implemented

1068-0314/16 $31.00 © 2016 IEEE
DOI 10.1109/DCC.2016.67

using the open-source Point Cloud Library (PCL) [6]. An example of this point cloud compression implementation is available at [7].

Separately from the work on processing point clouds, significant progress has been made over the past several decades on compressing images and video. Popular standards such as JPEG [8], H.264/AVC [9] and HEVC [10] are in widespread use today. These image and video coding standards also utilize block-based and/or hierarchical methods for coding pixels. Concepts from these image and video coders have also been used to compress point clouds. For example, H.264/AVC was used to compress point clouds in a teleoperation environment [11]. Significant improvements in coding efficiency were reported in [12], in which the PCL-based implementation of [5] was extended to use JPEG to code blocks of attribute values. As mentioned in the previously cited surveys, signal processing based approaches such as wavelets have also been used to compress point clouds. Another recent work that reports significant improvement over the PCL-based implementation of [5] is described in [13], in which a graph transform is applied to blocks of point cloud attributes.

In this paper, we build upon the idea of using approaches from image and video coding to improve the compression efficiency of point cloud coding systems. Specifically, we extend the concepts of block-based intra prediction and shape-adaptive transforms for compressing attributes from unstructured point clouds. In Section 2, we describe how point clouds are preprocessed and partitioned to comprise 3-D blocks of points on a uniform grid. Section 3 describes the 3-D block-based intra prediction of point cloud attributes. In Section 4, we introduce a modified shape-adaptive discrete cosine transform, and we describe an existing graph transform that is used for benchmarking. Section 5 presents experimental results, followed by a summary and conclusions in Section 6.

2. Point cloud preprocessing and block partitioning

Sometimes, point clouds are already arranged in a format that is amenable to block processing. For example, the graph transform in [13] operates on point clouds that are generated using the sparse voxelization approach described in [14]. The data in these point clouds are already arranged on a 3-D grid where each direction has dimensions 2^j with j being a level within a voxel hierarchy, and the points in each hierarchy level have integer coordinates. Partitioning such a point cloud into blocks, where the points are already arranged on a hierarchical integer grid, is straightforward. In general, however, point clouds captured using other techniques may have floating-point coordinate positions, not necessarily arranged on a grid.

In order to be able to process point clouds without constraint to any specific capture technique, we preprocess the original point cloud data so the points are located on a uniform grid. This preprocessing can also serve as a form of downsampling. This process is shown in Figure 1.

The first step of preprocessing is decomposing the point cloud to an octree representation. Related works using this kind of octree decomposition, implemented using the PCL software [6], can be found in [5] and [12]. Given an minimum octree resolution r, the point cloud is partitioned into into octree nodes. If a node contains

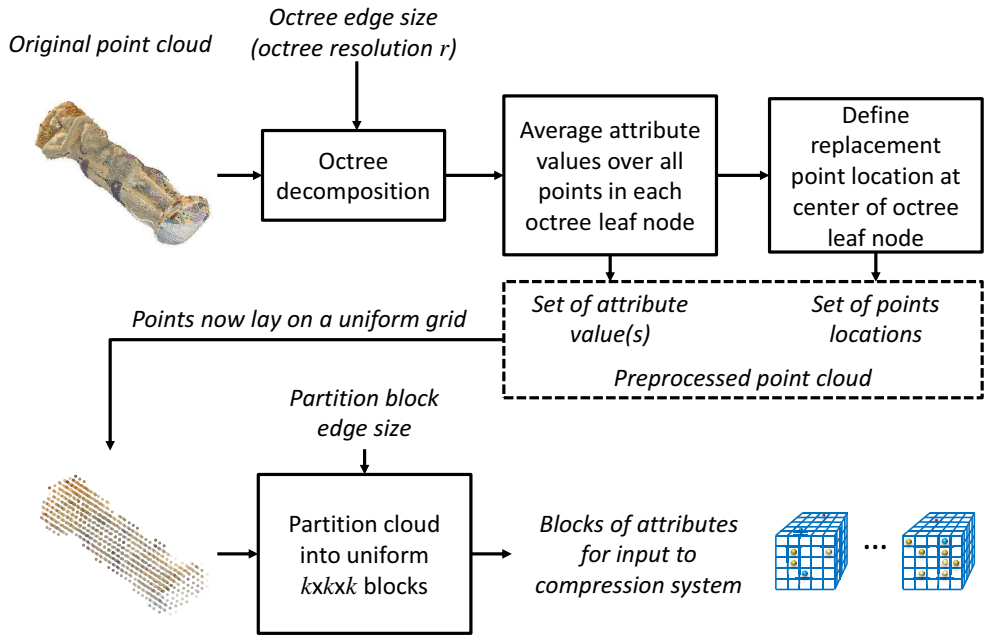


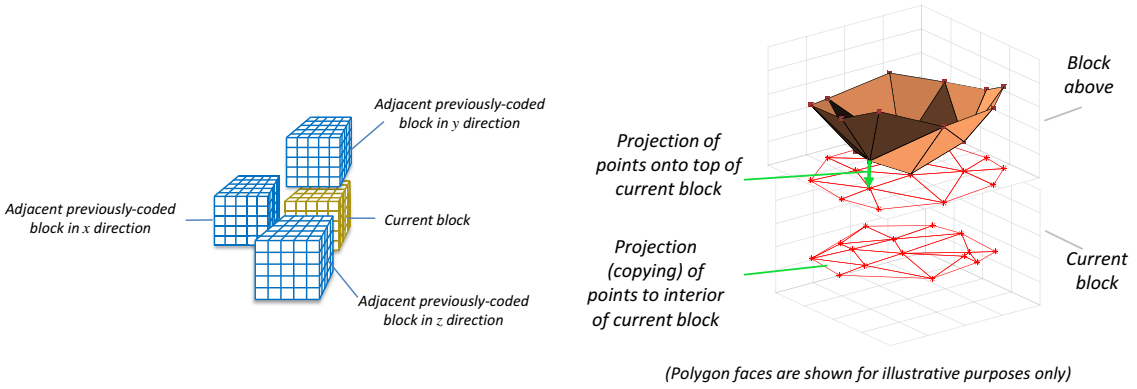
Figure 1: Preprocessing and block partitioning

no points, it is removed from the octree. If a node contains one or more points, it is further partitioned. This process continues until the size, or edge length of a leaf node reaches the minimum octree resolution r.

Each leaf node corresponds to a point output by this preprocessing step. The location of the output point is set to the geometric center of the leaf node, and the value of any attribute associated with the output point is set to the average value of the one or more points contained within the leaf node. This process ensures that the points output by the preprocessing step are located on a 3-D grid having resolution r.

Now that the points lie on a uniform grid, the region encompassing the set of points is partitioned into 3-D blocks of size $k \times k \times k$. A block contains k^3 positions or elements, however, many of these positions will be empty unless the point cloud happens to contain points at every possible position in each block.

At this stage, the difference between these 3-D point cloud blocks and 2-D blocks of pixels from image processing becomes apparent. In traditional image processing, all elements of a 2-D block correspond to pixel positions present in an image. In other words, the block is fully occupied. In the block-based point cloud processing in this paper, the 3-D blocks are not necessarily fully occupied. They can contain between 1 and k^3 elements. Tools such as intra prediction and block-based transforms used for image and video coding therefore cannot be directly applied to these 3-D blocks; modifications are needed to accommodate the empty positions.



(a) Available prediction directions (b) Multivariate interpolation/projection

Figure 2: 3-D block-based prediction

3. Intra prediction of 3-D point cloud blocks

Using prediction among blocks to reduce redundancy is a common technique in current coding standards such as H.264/AVC and HEVC. Neighboring decoded blocks are used to predict pixels in the current block, and then the prediction error or residuals are optionally transformed and then are coded in a bit-stream. For this paper, we will introduce a block prediction scheme for proof-of-concept purposes using a low-complexity prediction architecture in which the prediction is available from three directions. As shown in Figure 2a, points in the current block can be predicted from points contained in non-empty neighboring blocks, when adjacent neighboring blocks are available. The point cloud encoder performs prediction in the x, y, and z directions and chooses the prediction direction that yields the least distortion. Coding the current block without prediction from neighboring blocks is also considered if it yields lower distortion. Therefore, the current block has the option of being coded with or without prediction.

As mentioned in Section 2, many of the k^3 positions in a blocks may not be occupied by points. Moreover, points within a block may not necessarily be located along the edges of the block. The intra prediction techniques of H.264/AVC and HEVC use pixels along the neighboring blocks' boundaries to compute predictions for the current block. For our 3-D point cloud block prediction method, we will use multivariate interpolation/extrapolation to compute a projection of the attribute values in the neighboring block onto the adjacent edge plane of the current block. An example of this step is shown in Figure 2b. Here, data from known points are used to compute an interpolation or prediction located at an arbitrary point, in this case, along the boundary between the previous block and the current block. In our case, suppose the block above the current block contains a set of point locations $\boldsymbol{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_N\}$, with the points having associated attribute values $\boldsymbol{A} = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_N\}$. Given a point location along the boundary $\boldsymbol{p}_{\mathbf{boundary}}$, the prediction takes the form

$$\boldsymbol{a}_{\mathbf{boundary}} = f(\boldsymbol{P}, \boldsymbol{A}, \boldsymbol{p}_{\mathbf{boundary}}) \tag{1}$$

where $\boldsymbol{a}_{\mathbf{boundary}}$ is the predicted value of the attribute at the boundary. For this paper, we use nearest-neighbor interpolation/extrapolation, which reduces complexity and simplifies the handling of degenerate cases in which the neighboring block contains only one or two points, or when all the points in the neighboring block are aligned on a plane perpendicular to the projection plane.

Once the attribute values along the boundary plane are estimated, these values are then projected or replicated into the current block along the direction of prediction, similar to how prediction values are projected into the current block for the directional intra prediction used in standards such as H.264/AVC and HEVC. These projected and replicated values are then used to predict attributes for points present in the current block. For example, if the neighboring block in the y direction is used for prediction, then the set of points along the boundary $\boldsymbol{p}_{\mathbf{boundary}}$ are indexed in two dimensions, i.e. $p(x, z)$, and the attribute for a point the current block $p_{\text{curr}}(x, y, z)$ is predicted using $a_{\text{boundary}}(x, z)$ for all values of y.

4. Transforms for 3-D block data

After the prediction process, a 3-D block containing prediction residuals for each point in the current block, or the current block itself if it yields lower coding distortion, is ready to be transformed. As was the case for the prediction process, not all the positions in the block may be occupied by a point. The transform must therefore be designed so it will work on these potentially sparse blocks. We consider two types of transforms: a new variant of the shape-adaptive discrete cosine transform (SA-DCT) designed for 3-D point cloud attribute compression, and a 3-D graph transform.

4.1 Modified shape-adaptive DCT

The shape-adaptive DCT (SA-DCT) [15] is a well-known transform designed to code arbitrarily shaped regions in images. A region is defined by a contour, e.g. around a foreground region of an image. All the pixels inside the region are shifted and then transformed in two dimensions using orthogonal DCTs of varying lengths. The contour positions and quantized transform coefficients are then signaled in a bit-stream. For our 3-D point cloud application, we extend this concept to treat the presence of points in a 3-D block as a "region" to be coded, and positions in the block that do not contain points are considered as being outside the region. For the attribute coding application presented in this paper, the point positions are already available at the decoder irrespective of what kind of transform is used. Since the modified SA-DCT regions are defined by the point locations and not by the attribute values of the points, there is no need to perform operations like foreground and background segmentation and coding of contours as is typically done when the SA-DCT is used for image coding.

The modified SA-DCT process is shown in Figure 3. Given a 3-D block of attribute values or prediction residual values, the points present in the block are shifted line by line along dimension 1 toward the border so that there are no empty positions in the block along that border, except for empty lines. One-dimensional DCTs are applied along each occupied column of data along dimension 1, starting at the block

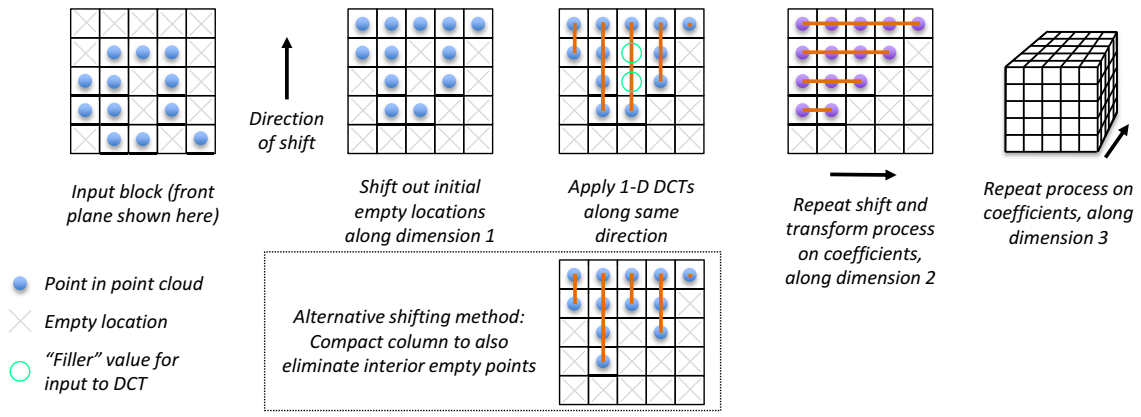


Figure 3: Modified shape-adaptive DCT

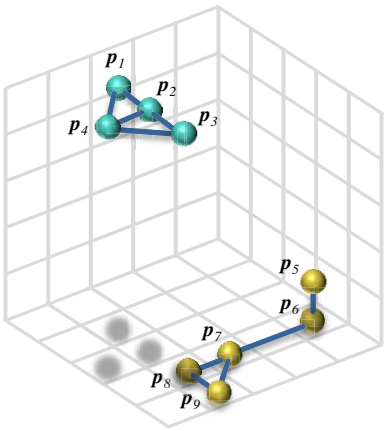
border and ending at the last data point present in the block for each column. If there are empty positions between the first and last points in the column, we insert filler values, e.g. zero. An alternative method that will be considered for future work would be to shift the remaining data in the column into those empty positions, thus reducing the lengths of the DCTs.

After the DCTs are applied along dimension 1, the shift and transform process is repeated on the transform coefficients along dimension 2. Finally, the process is applied along the third dimension, resulting in one DC and one or more AC coefficients. Compression is achieved by quantizing the coefficients.

4.2 3-D graph transform

One benchmark transform on 3-D blocks of attributes uses a graph transform, which draws upon the research performed on graph signal processing [16]. Because our point cloud is now partitioned into 3-D blocks, we can adapt the graph transform described in [13] into our framework by applying the graph transform on each block. For full details on how the graph transform is implemented, the reader is referred to [13].

The basic idea behind the graph transform is illustrated in Figure 4. A graph is formed by connecting adjacent points present in the 3-D block. Two points $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ are adjacent if they are at most one position apart in any dimension. Graph weights w_{ij} are assigned to each connection between points $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$, also known as a graph edge. The weights of a graph edge are inversely proportional to the distance between the two connected points. As described in [13], an adjacency matrix $\boldsymbol{A}$ is populated by the weights, from which a graph Laplacian matrix $\boldsymbol{Q}$ is computed. The eigenvector matrix of $\boldsymbol{Q}$ is used as a transform for the attribute values. After the transform is applied, each connected sub-graph has the equivalent of one DC coefficient and one or more AC coefficients. In contrast to the modified SA-DCT which always produces only one DC coefficient, the graph transform method will generate one DC coefficient for every disjoint connected set of points in the block, and each DC coefficient will have a set of corresponding AC coefficients. In the example of Figure 4, the graph is composed of two disjoint sub-graphs, so the resulting graph transform will produce two DC coefficients and two corresponding sets of AC coefficients.



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

Figure 4: Example of block containing two disjoint sets of connected points, and the corresponding adjacency graph

5. Experimental results

A flowchart showing the preprocessing and coding steps used for experiments is shown in Figure 5. The input point cloud is preprocessed as described in Section 2 to generate a point cloud situated on a uniform grid. Next, the block partitioning, intra prediction, and transform steps are applied. We assume a sign-magnitude coder will be used to code the transform coefficients, so the initial results presented in this paper report the entropies of the data to be signaled vs. PSNR metrics computed against the uniform-grid point cloud input to the block processing system. A uniform quantizer is used to quantize the transform coefficients, with a fixed step size set to determine the amount of compression. For the graph transform experiment, we report additional results using bit rates achieved when an arithmetic coder is used to code the quantized coefficients.

The input point cloud used for experiments is the *Statue_Klimt_PointCloud.ply* unstructured point cloud described in [17] and shown in Figure 1. This point cloud contains approximately 500k points, with each point having a floating-point (x, y, z) location with 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 used in all experiments. The octree resolution used for preprocessing is $r = 1.0$. Coding performance as luminance PSNR vs. the entropy in bits needed to represent the attribute, assuming a sign-magnitude representation of the coefficients, is shown in Figure 6. Coding performance results for block sizes $k = 5$ and $k = 10$ are shown, along with results for when one large block is used to contain the entire point cloud. Generally, the modified SA-DCT outperformed the graph transform approach. Part of this difference in performance may be because the graph transform typically produced a DC coefficient that was significantly larger than the AC coefficients, and for sparse blocks containing few connected points, more DC coefficients are present in each block when the graph transform is used. More bits are therefore needed to represent that larger number of DC coefficients.

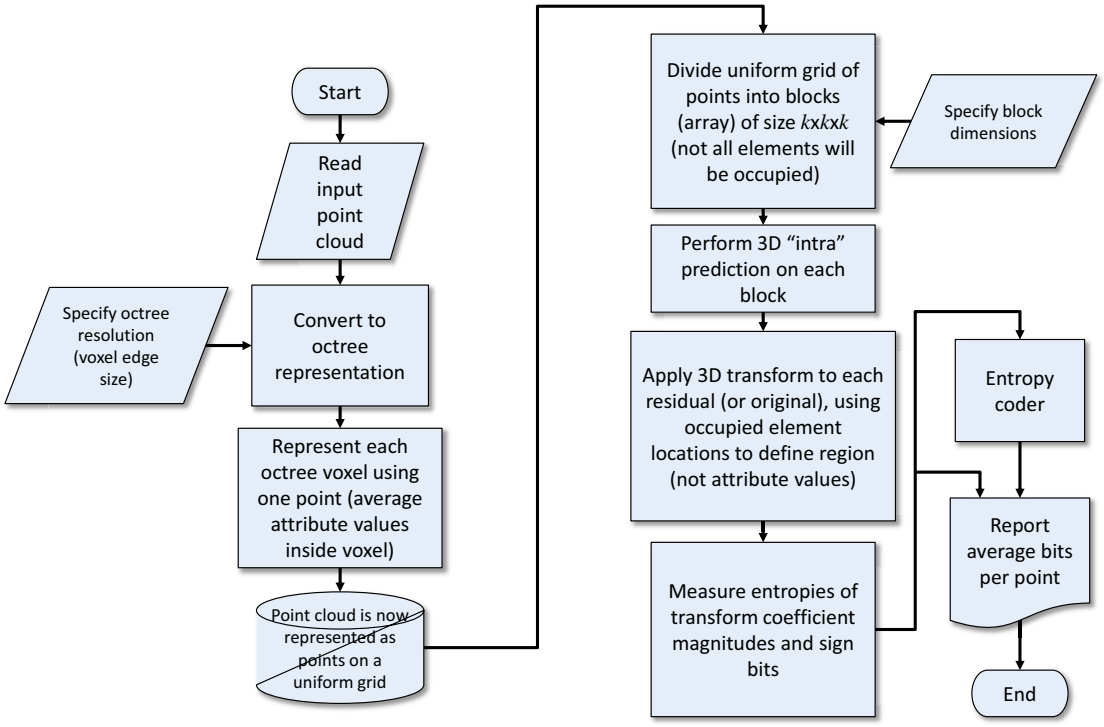


Figure 5: Preprocessing and coding steps used for experiments

The graph transform approach in [13] uses a custom-designed arithmetic coder to code the transform coefficients, where the eigenvalues associated with the eigenvectors of the graph transform matrix are used to compute the diversity parameters of the Laplacian distributions used to derive the probability tables. We also apply a Laplacian-based arithmetic encoder to code the graph transform coefficients, and we compare the arithmetic coding results to the estimated entropies in Figure 7. As expected, the arithmetic coding results are closer to the entropies as the block size becomes larger, and when the entire point cloud is treated as one large block, the arithmetic coding performance is almost the same as the entropy.

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. Point clouds are first preprocessed so the points are aligned to a uniform grid, and then the grid is partitioned into 3-D blocks. Unlike image/video processing in which all points in a 2-D block correspond to a pixel position, our 3-D blocks are not necessarily fully occupied by points. After performing 3-D block-based intra prediction, we transform and quantize the resulting data. A modification of the shape-adaptive DCT is presented, which is used to transform these blocks of data that can have missing points. Compression performance using the modified 3-D shape-adaptive DCT and 3-D graph transform benchmark are presented, along with results comparing entropy performance to arithmetic coding performance for the graph transform method. Future work will focus on

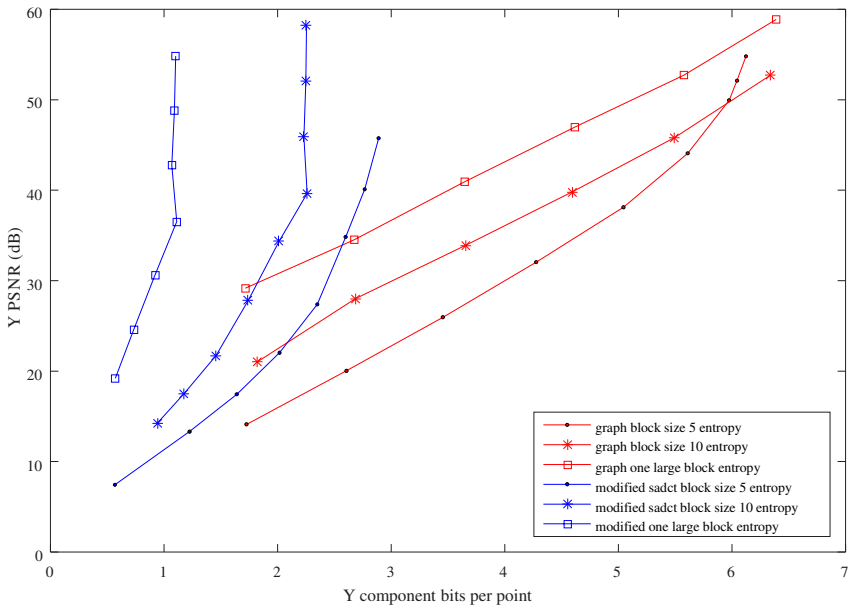


Figure 6: Performance of block-based point cloud compression method

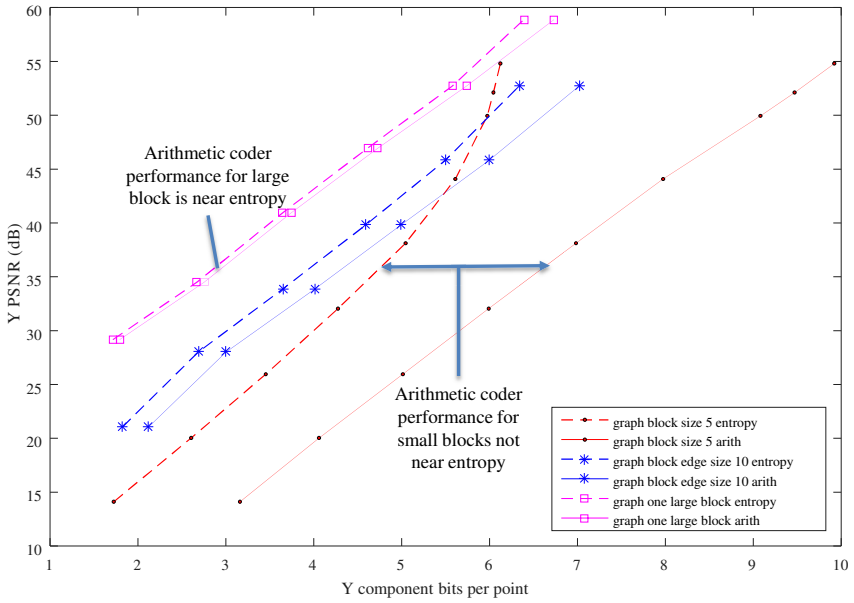


Figure 7: Entropy and arithmetic coding performance with graph transform approach

reducing the number of DC coefficients generated by very sparse blocks, developing entropy coders suitable for coding the modified shape-adaptive DCT coefficients, and incorporating more concepts from state of the art codec such as HEVC into the point cloud compression system.

7. References

[1] J. Peng, C.-S. 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] P. Alliez and C. Gotsman, "Recent advances in compression of 3D meshes," in *In Advances in Multiresolution for Geometric Modelling*. Springer-Verlag, 2003, pp. 3–26.

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

[4] J. 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. 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] R. B. Rusu and S. 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, W.-J. 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] X. Xu, B. Cizmeci, and E. 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," ISO/IEC JTC1/SC29/WG11 *Coding of Moving Pictures and Audio*, Geneva, Switzerland, document MPEG2015/m35910, Feb. 2015.

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

[14] C. T. Loop, C. Zhang, and Z. 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.

[15] 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.

[16] D. Shuman, S. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *Signal Processing Magazine, IEEE*, vol. 30, no. 3, pp. 83–98, May 2013.

[17] V. Scurtu, C. Tulvan, A. Gabrielli, and M. Preda, "Reconstruction platform and datasets for 3D pointcloud compression," ISO/IEC JTC1/SC29/WG11 *Coding of Moving Pictures and Audio*, Warsaw, Poland, document MPEG2015/m36523, 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.