Title

PATCH-AWARE AVERAGING FILTER FOR SCALING IN POINT CLOUD COMPRESSION

Authors

Cao, Keming
Xu, Yi
Cosman, Pamela C

Publication Date

2018-11-29

DOI

10.1109/globalsip.2018.8646392

Peer reviewed

PATCH-AWARE AVERAGING FILTER FOR SCALING IN POINT CLOUD COMPRESSION

Keming Cao[⋆] *Yi Xu*[†] *Pamela C. Cosman*[⋆]

[⋆] Department of Electrical and Computer Engineering
UC San Diego, La Jolla, CA 92093-0407

[†] Owlii Inc.
Beijing, China

ABSTRACT

With the development of augmented reality, the delivery and storage of 3D content have become an important research area. Among the proposals for point cloud compression collected by MPEG, Apple's Test Model Category 2 (TMC2) achieves the highest quality for 3D sequences under a bitrate constraint. However, the TMC2 framework is not spatially scalable. In this paper, we add interpolation components which make TMC2 suitable for flexible resolution. We apply a patch-aware averaging filter to eliminate most outliers which result from the interpolation. Experimental results show that our method performs well both on objective evaluation and visual quality.

Index Terms— Interpolation, Point Cloud Compression, Outlier Elimination, Patch-aware Averaging Filter, Augmented Reality

1. INTRODUCTION

Virtual reality (VR) and augmented reality (AR) are important future directions of interaction with content and the real world, due to the better experience they could provide compared to traditional 2D media content. The development of depth sensors, such as Kinect from Microsoft and RealSense from Intel, leads to an increase of 3D data processing applications. This in turn makes the storage for huge amounts of 3D data a problem.

There are two main representations for 3D data, mesh and point cloud. For point cloud compression, since the points are not connected as they are in a mesh, the spatial organization of points should be built so as to encode the points efficiently. Octree is applied in [1, 2] to progressively compress point clouds, which makes a partition of three dimensional space by dividing it into octants recursively like a tree structure with eight children nodes for each parent node. Hierarchical clustering of points can generate Level Of Detail which is progressively compressed in [3]. MPEG hosted a call for proposals [4] and picked out three methods as winners for three different categories: static model, dynamic sequence and dynamic acquisition. TMC2 from Apple Inc. [5] achieves the best subjective and objective quality under given target bitrates for the dynamic sequence category. Its core idea is to project points, both their geometry coordinates and attributes, to 2D and convert the 3D sequence to a 2D sequence. Then any present video codec, such as ffmpeg and HM (Test Model for HEVC) could be utilized to compress the 2D sequence. In their framework, after the resolution of the 2D sequence is fixed as an initial parameter, any scale-up or scale-down of the 2D sequence will create outlier points when projected back to 3D space. However, there is no prior work dedicated to removing those outlier points for point cloud compression. In this paper, we add interpolation modules which makes TMC2 suitable for flexible resolution. A patch-aware averaging filter is proposed to eliminate the outlier points which result from scaling-down and scaling-up. Experimental results show that our method performs well both on objective evaluation and visual quality.

This paper is structured as follows. Sec. 2 briefly describes the framework of TMC2 and Sec. 3 introduces the details of our proposal. Sec. 4 shows experimental results and conclusions.

2. RELATED WORK

The framework of TMC2 is shown in Fig. 1. Here, the color information assigned for each point, called texture information, represents an attribute of a point. Any other kind of attribute, such as reflectance, could be processed in the same way. The framework of the encoder is divided into three parts: segmentation, packing and compression.

2.1. Segmentation

Given an input frame of a point cloud, before segmenting the points, TMC2 generates the normal direction for each point based on the underlying surface. Then, according to normal direction clusters, segmentation is applied to divide the point cloud into patches. The total number of patches is constrained within 256. Fig. 2 from [5] shows an example.

For each patch $\boldsymbol{p}_i$, $i \in \{1, 2, ..., N\}$ where N is the number of patches, one of 6 main directions ($\pm\boldsymbol{x}$ direction, $\pm\boldsymbol{y}$ direction and $\pm\boldsymbol{z}$ direction) is assigned as the principal direction based on the main clustered normal direction.

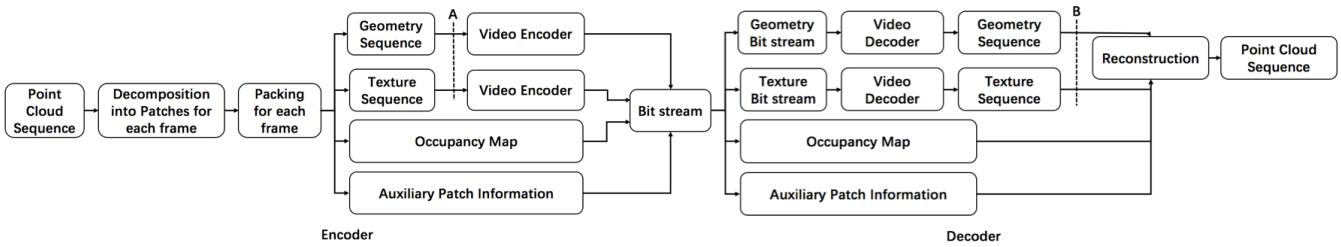

Fig. 1. Compression framework

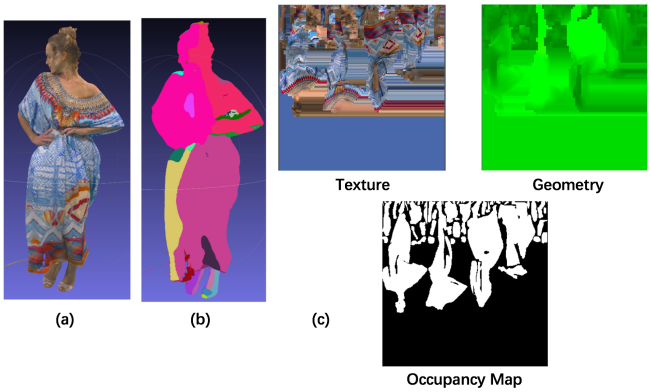

Fig. 2. (a) Original point cloud. (b) Segmentation result with different colors representing different patches (c) Examples for texture image, geometry image and occupancy map

2.2. Packing

After segmentation, projection is performed to convert each 3D patch into a 2D patch. For each point $\boldsymbol{v}$ in the 3D patch $\boldsymbol{p}_i$, the coordinate of $\boldsymbol{v}$ is projected onto the 2D patch $p_i^{geometry}$ while the color information is projected onto $p_i^{texture}$. The projection direction is determined by the principal direction of this 3D patch. The same location of a pixel in $p_i^{geometry}$ and $p_i^{texture}$ denotes one same 3D point. Then packing fits all 2D texture patches into an image $I_{texture}$, and all geometry patches into an image $I_{geometry}$ with the same pre-fixed size l_x by l_y. Packing images from all 3D frames will form two 2D video sequences, one for geometry and the other for texture. Geometry and texture sequences are in YUV format but only the Y channel is occupied for the geometry sequence because for 3D point (x, y, z), z is stored in Y channel at pixel position (x, y).

Apart from geometry and texture, occupancy maps are also generated for each frame where 0 denotes not occupied and 1 denotes occupied. Occupancy maps are losslessly compressed with arithmetic coding. Fig. 2 (c) shows image examples for texture, geometry and occupancy map.

2.3. Compression and Decompression

Geometry and texture sequences could be encoded with any 2D video codec, such as ffmpeg or HM. Combining all information together, we get the final compressed bit stream. The decoder shown in Fig. 1 is a reversed encoder. First, the bit stream is separated into the geometry, texture, occupancy map and patch information. Then, we decode the geometry and texture bit streams, and project 2D patches back to 3D according to patch information and occupancy maps.

3. PROPOSED METHOD

In the original TMC2 framework, the image size, l_x and l_y, is fixed all through the compression, which could lead to less flexibility when compressing the dynamic sequence, especially when a target bitrate constraint exists. Although the geometry and texture sequences could be scaled down at point A in Fig. 1 at the encoder and be scaled up at point B at the decoder, interpolation for scaling tends to bring in noise at the edges of patches, especially for the geometry sequence. An example of noise in the geometry image after scaling is shown in Fig. 3. The video content of the geometry sequence is not like normal 2D videos, which have strong spatial correlation, because the geometry images are packed from different patches with different depths. It is also different from the content of the texture sequence because the noise of the color information is not that obvious compared to geometry noise. Geometry noise will easily turn out to be a large outlier when back-projected to 3D space. So our proposed method is targeted to process the geometry image so as to reduce the outliers.

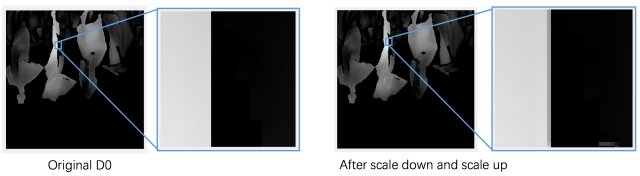

Fig. 3. Scaling noise is the middle strip in the right image

We add scaling modules at the encoder and decoder to improve flexibility for compression, however, we must perform

further processing to remove noise caused by scaling. Pixels in 2D geometry patch $p_i^{geometry}$ are divided into three sets. Set S_{n1}^i has pixels of chessboard distance 1 to pixels outside the patch, set S_{n2}^i has pixels of chessboard distance 2 to pixels outside patch and the remaining pixels of the patch, which are the interior pixels, are in set S_b^i. Pixel values in set S_{n1}^i are modified by a patch-aware averaging filter:

$$I'_{up}(x,y) = \begin{cases} \frac{1}{M} \sum_{m \in Q} I_{up}(x_m, y_m), & Q \neq \emptyset \\ I_{up}(x,y), & Q = \emptyset \end{cases} \quad (1)$$

where Q is the set of points that belong to set S_{n2}^i and have chessboard distance 1 to (x, y) and M is the size of Q. The left image in Fig. 4 shows examples. The filtered value for pixel A should be the average of pixels 1, 2, 3 and 4, for pixel B it should be pixel 5, and it is the average of pixels 6, 7, 8 for pixel C.

Here an averaging filter is chosen for simplicity. Other simple filters could also be applied, such as a median filter or weighted filter. Different filters are evaluated in Sec. 4.

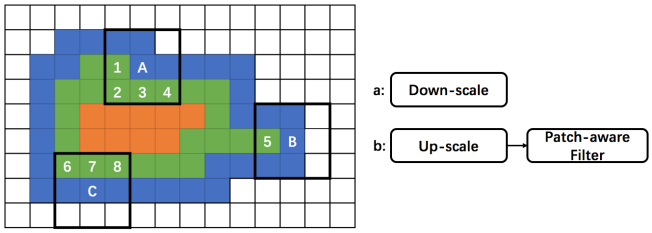

Fig. 4. Example for patch-aware averaging filter and proposed modules

Our modules added to the TMC2 framework are shown as 'a' in Fig. 4 to be inserted at point A in Fig. 1 and 'b' in Fig. 4 to be placed at point B in Fig. 1. Here, the down-scale module is a normal scaling component for any interpolation $method$, $I_{down} = \mathbf{Scale}(I, 1/r, method)$, where I is the input image, $r \geq 1$ is the scale factor and I_{down} is the down-scaled image I. The up-scale module is a normal scaling component for any interpolation $method$, $I_{up} = \mathbf{Scale}(I_{down}, r, method)$, and the size of I_{up} is equal to that of I. Two interpolation methods are evaluated in Sec. 4.

4. EXPERIMENT

4.1. Dataset

We choose four 3D dynamic sequences from Owlii Inc. as our dataset: *basketball*, *dancer*, *model* and *exercise*. Each sequence has 300 frames with around 2.5 million points per frame with range 1024 for x, y and z directions.

4.2. Evaluation Metric

For 2D evaluation, we choose the PSNR of the occupied pixels denoted in the occupancy map as our evaluation metric. However, MSE is chosen for 3D because it is hard to define a fixed maximum possible value for different 3D sequences. Given input reference point cloud A and processed point cloud B, first, we determine an error vector $\boldsymbol{e}_{AB}(i,j)$ from point $\boldsymbol{p}_i$ in A to the closest point $\boldsymbol{p}_j$ in B. Then we compute the MSE_{AB} as $\frac{1}{N_A} \sum_{\boldsymbol{p}_i \in A} |\boldsymbol{e}_{AB}(i,j)|^2$ where N_A is the number of points in A. The error vector from B to A, $\boldsymbol{e}_{BA}(j,i)$, is generated in a similar way which leads to MSE_{BA} as $\frac{1}{N_B} \sum_{\boldsymbol{p}_j \in B} |\boldsymbol{e}_{BA}(j,i)|^2$ and N_B is the total number of points in B. The final MSE_{final} is set to $\mathrm{MSE}_{final} = \max(\mathrm{MSE}_{AB}, \mathrm{MSE}_{BA})$.

4.3. Evaluation

For each test sequence, we randomly pick three frames to evaluate with both the 2D metric and 3D metric. Six scale factors, 1.25, 1.5, 1.75, 2.0, 2.25, 2.5, are chosen with four different choices for interpolation:

NN2NN: Nearest neighbor scaling at the encoder and decoder.

NN2B: Nearest neighbor scaling at the encoder and bicubic scaling at the decoder.

B2NN: Bicubic scaling at the encoder and nearest neighbor scaling at the decoder.

B2B: Bicubic scaling at the encoder and decoder.

First, patch-aware averaging filter is evaluated. For 2D PSNR, we have $\mathrm{PSNR}_1 = \mathrm{PSNR}(I, I_{up})$ and $\mathrm{PSNR}_2 = \mathrm{PSNR}(I, I'_{up})$. I is the original image, I_{up} is the image after scaling down and scaling up and I'_{up} is I_{up} processed with the patch-aware averaging filter. Table 1 shows $\Delta_{PSNR} = \mathrm{PSNR}_2 - \mathrm{PSNR}_1$ with different interpolation choices. We see that almost for every scale factor and every sequence, the patch-aware averaging filter produces a quality improvement. For 3D MSE, we evaluate the method in a similar way. MSE_1 is computed between the input point cloud and output point cloud from compression and de-compression without the patch-aware averaging filter. MSE_2 uses the output point cloud after the patch-aware averaging filter. Five quantization parameters (QP) are chosen: 0, 10, 20, 30 and 40. The scale factors are the same as for the PSNR evaluation. Then $\Delta_{MSE} = \mathrm{MSE}_2 - \mathrm{MSE}_1$. In Fig. 5, Δ_{MSE} vs. QP curves are plotted for different interpolation choices and different r. Only plots for sequence *dancer* are shown to save space. Almost all curves are below the x axis which means the patch-aware averaging filter produces a lower MSE.

Three filter types, patch-aware averaging filter (A), patch-aware median filter (M) and patch-aware weighted filters (W) of 3×3 Gaussian kernel, with two interpolation choices *NN2NN* and *NN2B* are compared. $A - M$ is the difference of MSE between A and M and other notations use similar defi-

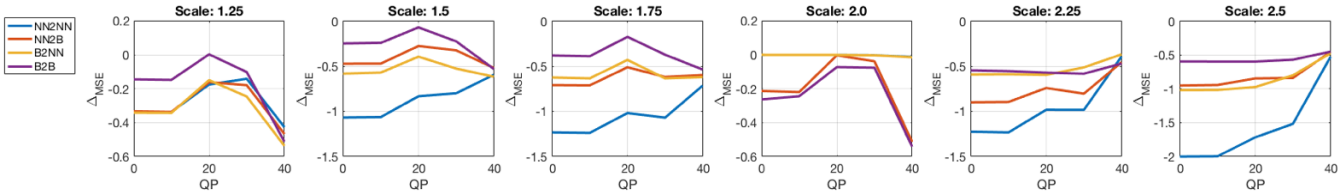

Fig. 5. Δ_{MSE} vs. QP for different scale factors of sequence *dancer*

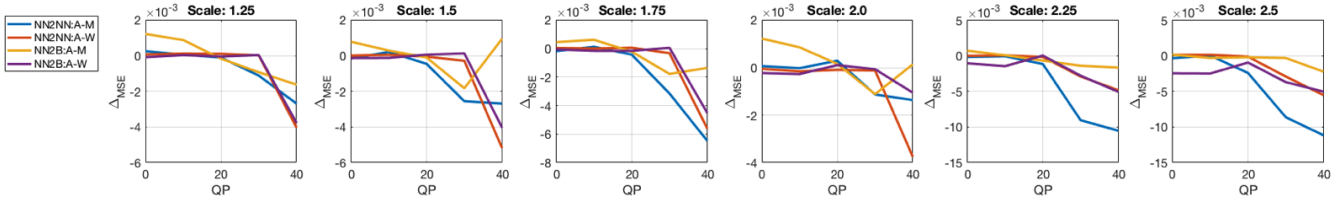

Fig. 6. Δ_{MSE} vs. QP for different scale factors of sequence *dancer* for filter comparisons

nitions. The curves for different QPs and r are shown in Fig. 6 for *dancer*. In the plots, all three filter types have similar performance because the Δ_{MSE} are all within 0.01 and the averaging filter is slightly better than the other two especially for QP $\geq$ 20. When QP $\leq$ 20 and $r \leq 2.0$ for *NN2B*, the yellow curves are above x axis which means median filtering is a little better than averaging filtering.

Table 1. Δ_{PSNR}

Factors	1.25	1.5	1.75	2.0	2.25	2.5
NN2NN:						
dancer	5.463	5.235	7.230	-0.008	7.334	7.183
basketball	4.659	6.800	7.802	0.012	7.415	8.166
model	5.299	5.600	7.017	-0.131	6.455	7.610
exercise	6.419	6.389	7.838	-0.198	8.119	8.149
NN2B:						
dancer	7.991	8.200	7.587	4.194	5.474	4.576
basketball	8.417	9.006	8.281	4.928	5.819	4.663
model	8.225	8.238	7.293	4.497	5.109	4.663
exercise	9.684	9.256	8.287	5.304	5.721	5.000
B2NN:						
dancer	6.817	6.161	6.690	0.042	5.253	4.693
basketball	7.482	7.534	7.625	0.070	5.137	4.732
model	6.485	6.403	6.893	-0.094	4.388	4.932
exercise	7.343	7.191	8.021	-0.065	5.600	5.306
B2B:						
dancer	6.924	7.647	7.506	7.366	5.720	4.704
basketball	7.572	8.403	8.080	8.363	6.135	4.912
model	6.687	7.449	7.161	7.301	5.402	4.755
exercise	7.581	8.455	8.194	8.485	5.801	5.245

To choose which interpolation can generate the best quality, following evaluation method is applied. For each sequence and interpolation method, $mse_{QP,1}$ is first subtracted from the MSE value $mse_{QP,r}$ corresponding to different scale factors for the same QP. The resulting values are normalized to $[0, 1]$ across all interpolation methods. Lining up normalized values for all QPs and all factors, we calculate the mean value for four interpolation choices shown in Table 2.

Table 2. MEAN$_{MSE}$

Interpolation	NN2NN	NN2B	B2NN	B2B
dancer	**0.170**	0.343	0.736	0.593
basketball	**0.260**	0.390	0.747	0.516
model	**0.266**	0.321	0.751	0.484
exercise	**0.227**	0.474	0.742	0.564

A lower mean value denotes better MSE. We notice that the mean value for *NN2NN* is the lowest among the four choices for every sequence, so *NN2NN* is the best combination among the four interpolation choices.

For visual quality, Fig. 7 shows examples without and with the patch-aware averaging filter. Our filter eliminates the outliers and greatly improves visual quality.

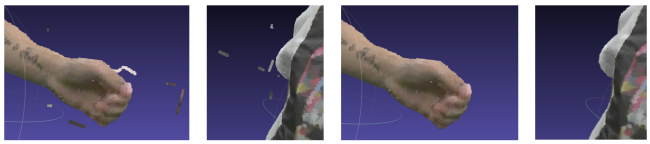

Fig. 7. The left two images are without filtering and the right two are with filtering.

Conclusion: In this paper, we add scaling modules to the original TMC2 framework to improve compression flexibility. A patch-aware averaging filter is proposed to remove most outlier points caused by scaling. The averaging filter generates slightly lower MSE than the median filter and a weighted filter. The experimental results on 2D PSNR and 3D MSE show that our method performs well both on objective evaluation and on subjective visual quality. Among the four different interpolation choices, *NN2NN* achieves the best performance in terms of MSE.

Acknowledgment: This research was supported in part by NSF grant SCH-1522125 and by Owlii Inc.

5. REFERENCES

[1] R. Schnabel and R. Klein, "Octree-based point-cloud compression.," *Spbg*, vol. 6, pp. 111–120, 2006.

[2] J. Kammerl, N. Blodow, R. Bogdan 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*. IEEE, 2012, pp. 778–785.

[3] Y. Fan, Y. Huang, and J. Peng, "Point cloud compression based on hierarchical point clustering," in *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2013 Asia-Pacific*. IEEE, 2013, pp. 1–7.

[4] "Call for proposals for point cloud compression v2," ISO/IEC JTC1/SC29/WG11, Hobart, Australia, April, 2017.

[5] K. Mammou, A. M. Tourapis, D. Singer, and Y. Su, "Video-based and hierarchical approaches point cloud compression," ISO/IEC JTC1/SC29/WG11, Macau, China, October, 2017.