Distance-Based Probability Model for Octree Coding

Ricardo L. de Queiroz, *Fellow, IEEE*, Diogo C. Garcia, *Member, IEEE*, Philip A. Chou, *Fellow, IEEE*, and Dinei A. Florencio, *Fellow, IEEE*

***Abstract*—We present a context-driven method to encode nodes of an octree, which is typically used to encode the point-cloud geometry. Instead of using one bit per node of the tree, the context allows for deriving probabilities for that node based on distances of the actual voxel to the voxels in a reference point cloud. Accurate probabilities of the node state allow for the use of an arithmetic coder to reduce the bit rate. Results point to potentially large reductions in rate if there is a good model from which to derive the context, i.e., one can get large reductions if the reference-cloud geometry is close enough to the one being encoded.**

***Index Terms*—Compression for augmented reality, octree representation, point-cloud compression, 3D compression.**

I. Introduction

POINT clouds are used for the three-dimensional (3-D) depiction of scenes and objects in many applications. We are concerned with telepresence systems that involve a 3-D representation of a person or object and its rendering at a remote location with augmented reality techniques, e.g., in holoportation [1], [2]. Point clouds are typically represented using arbitrary positions in space. We, however, are interested in voxelized point clouds, where the points are made to adhere to a uniform grid, thus forming cubic volumetric elements (voxels) that may or may not be occupied. An occupied voxel usually has attributes, such as colors and normals, but we are not considering them in this letter. We are only concerned with the geometry, i.e., with the position of the points in the cloud, effectively with the location of where the voxels are occupied. Octrees are quite popular to encode the point-cloud geometry [3]–[5] and have also been used in recent works dealing with point-cloud compression [6]–[9]. The use of contexts in octree coding has also been explored in [10] looking for a correlation among the bytes in the encoded bitstream.

Manuscript received January 29, 2018; revised March 19, 2018; accepted March 30, 2018. Date of publication April 5, 2018; date of current version April 23, 2018. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Matteo Naccari. (*Corresponding author: Ricardo L. de Queiroz.)*

R. L. de Queiroz is with the Department of Computer Science, Universidade de Brasilia, Brasilia 70910-900, Brazil (e-mail: queiroz@ieee.org).

D. C. Garcia is with Gama Engineering College, Universidade de Brasilia, Brasilia 70910-900, Brazil (e-mail: diogo@divp.org).

P. A. Chou is with 8i Labs, Bellevue, WA 98052 USA (e-mail: pachou@ieee.org).

D. A. Florencio is with Microsoft Research Lab, Redmond, WA 98052 USA (e-mail: dinei@microsoft.com).

Color versions of one or more of the figures in this letter are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LSP.2018.2823701

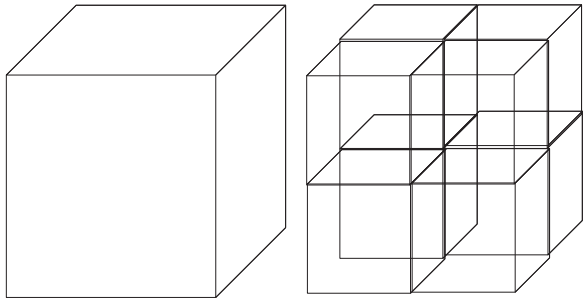

Fig. 1. In a particular level, a unit cube is divided into eight subcubes by splitting each dimension into two halves. Each subcube is further subdivided in the same manner.

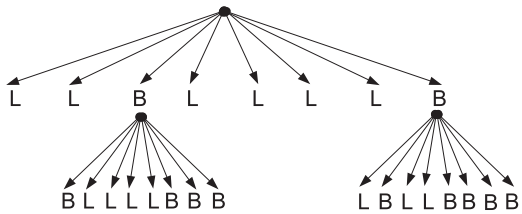

Fig. 2. Spanning the space, traversing the octree. Unoccupied subcubes are made leaves of the tree and the occupied ones are further subdivided.

II. Voxel Position Encoding

The points are discretized into a grid of regular volumetric elements or voxels. Let the voxel space comprises a cube of $M \times M \times M$ homogeneous voxels, where $M = 2^D$ and D is the depth of the decomposition. Each of the 2^{3D} voxels may be occupied or not. Let $w(i, j, k)$ represent the binaryoccupation state of a given voxel (occupied or not) in the region of interest, where (i, j, k) is the address of such a voxel in the discrete grid. Typically, less than 2% of the voxels are actually occupied, such that it is easier to record a voxelized point cloud as a list of occupied voxels $\{\nu_i\}$, where each voxel is described by a triplet as $\nu_i = [x_i, y_i, z_i]$, where x_i, y_i, and z_i are the Cartesian coordinates of the voxel position in the grid. In other words, one can either describe the state of all voxels, or simply list the occupied ones.

The octree is an efficient method to encode the geometry. In the first level of the octree, the voxel space is partitioned into eight smaller cubes of dimensions $M/2 \times M/2 \times M/2$, as depicted in Fig. 1. If a smaller cube has at least one occupied voxel, it is flagged occupied, otherwise it is flagged empty. In order to build a tree (octree), as shown in Fig. 2, the occupied cube is marked as an internal or branch node B, while the empty one is marked as a leaf node L. In a second level, each

of the occupied cubes is further partitioned in exactly the same way, each generating eight cubes of dimensions $M/4 \times M/4 \times M/4$. If all cubes are split, we obtain 64 cubes of reduced dimensions. The process can be repeated for up to D levels, yielding 2^{3D} voxels in the last level, which is the entire space described by $w(i, j, k)$.

Let the voxel space be a cube of width W. It can be regarded as a voxel itself at the root node (level 0) of the octree, which we label as voxel $\nu_{0,0}$ and node $\theta_{0,0}$. It may be partitioned into eight cubes of width $W/2$, and each one can be considered as a voxel at level 1, $\nu_{1,k}$, with respective nodes $\theta_{1,k}$, $(k = 0, 1, \ldots, 7)$. At level ℓ, the space is partitioned into $2^{3\ell}$ cubic voxels of width $W/2^{\ell}$, out of which N_ℓ are occupied. Note that the number K_ℓ of tree nodes $\{\theta_{\ell,k}\}$ may be different, i.e., $K_\ell \geq N_\ell$, since the occupied voxels at a level represent the B nodes, and one needs to account for the L nodes, which are also present in the octree.

As in the example in Fig. 2, the sequence of $\{\theta_{\ell,k}\}$ as B or L labels completely describes the tree, thus the voxels positions. This is the information to be encoded and transmitted.

III. Context and Arithmetic Coding for Octrees

In the absence of further information, the probability of a node being a leaf or not is assumed the same $P(\theta_{\ell,k} = L) = P(\theta_{\ell,k} = B) = 0.5$ and octrees are invariably encoded using one bit per node. If, however, we can estimate these probabilities $[P(\theta_{\ell,k} = L) = 1 - P(\theta_{\ell,k} = B)]$ and use a perfect arithmetic coder, we may encode a leaf node with $-\log_2[P(\theta_{\ell,k} = L)]$ bits and a branch node with $-\log_2[P(\theta_{\ell,k} = B)]$ bits. Note that the higher the probability, the lower the rate. Hence, if we predict well the type of the node, we increase its probability and decrease the rate. In one extreme, if we have no model to predict θ being a leaf, for example, then $P(\theta = L) = P(\theta = B) = 0.5$ and the node is encoded with 1 b. On the other extreme, if there is an absolute certainty of the node being a leaf, $P(\theta = L) = 1$ and there is no need to spend any bits to encode it. If the model indicates that $P(\theta = L)$ is high, encoding the node as a leaf spends a very low rate. However, if it is actually an internal node $(\theta = B)$, then one will spend a very high rate to encode the node. Aggressive prediction leads to large gains if you make the right prediction, but also yields large losses if the prediction is incorrect.

Let the point-cloud frame be F and assume we use a reference frame $\hat{F}$, from which to derive the context. We adopt a distance-based probability estimation. We will refer to occupied voxels simply as voxels, unless marked otherwise. If F has voxels ν_{ij} and $\hat{F}$ has voxels $\hat{\nu}_{ij}$, we estimate the probabilities based on minimum distances in between the voxels of the actual and reference frames. Let

$$\delta_{\ell,k} = \min_i \text{dist}(\nu_{\ell,k}, \hat{\nu}_{\ell,i}) \tag{1}$$

where the voxel distance function can be Euclidean, for example. We, then, model the probabilities on how far the voxel in the actual frame is from any voxel in the reference frame, i.e.,

$$P(\theta_{\ell,j} = L) = f\left(\delta_{\ell,k}\right). \tag{2}$$

0	1	1	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0

(a)

1	0	0	1	4	8	13	17	20	25
1	0	0	1	2	5	8	10	13	18
2	1	0	0	1	2	4	5	8	13
5	2	1	0	0	1	1	2	5	10
8	5	2	1	0	0	0	1	4	8
13	8	5	2	1	1	0	1	2	4
18	13	8	5	4	1	0	0	1	1
25	18	13	10	5	2	1	0	0	0
32	25	20	13	8	5	2	1	1	0
41	34	25	18	13	8	5	4	2	1

(b)

B	A	A	B	D	F	H	I	K	L
B	A	A	B	C	E	F	G	H	J
C	B	A	A	B	C	D	E	F	H
E	C	B	A	A	B	B	C	E	G
F	E	C	B	A	A	A	B	D	F
H	F	E	C	B	B	A	B	C	D
J	H	F	E	D	B	A	A	B	B
L	J	H	G	E	C	B	A	A	A
M	L	K	H	F	E	C	B	B	A
O	N	L	J	H	F	E	D	C	B

(c)

Fig. 3. Context example in 2-D. (a) Map with the voxels in the reference frame. (b) Squared distances to occupied voxels in the reference frame. (c) Contexts assigned to each distance. Context "A" indicates the position of the reference voxels (distance 0).

Function f can be modeled or simply inferred and used as side information. $\delta_{\ell,k}$ is the context in which the probabilities are computed. What we need to compute is $P(\theta_{\ell,j} = L|\delta_{\ell,k} = d)$ for all the significant values of d and ℓ. If one assumes the reference to be similar to the actual frame, the idea is that the closer the voxel to the reference position, the more likely the voxel is to be occupied. A 2-D depiction of this context is illustrated in Fig. 3. Fig. 3(a) is the 2-D map of the reference voxels. Fig. 3(b) shows the distances (squared for clarity) to each position to the closest occupied voxel in the reference frame. From the distances, we assign the contexts, which are depicted in Fig. 3(c), for example, assigning one context to each distance value.

The model can be used in a few applications including compression or super-resolution of point clouds.

A. Examples on Compressing Dynamic Point Clouds

Dynamic point clouds are videos of point clouds, i.e., point-cloud frames displayed in sequence. Hence, while encoding

TABLE I
COMPARISON AMONG METHODS FOR COMPRESSING THE GEOMETRY OF POINT-CLOUD SEQUENCES

Sequence	No model	[10]	Method I	Method II
Andrew	2.58	1.69	1.88	1.73
David	2.62	1.83	2.41	1.87
Man	3.16	2.29	2.71	1.94
Phil	2.64	1.88	2.31	1.88
Ricardo	2.92	2.22	2.60	2.25
Sarah	2.61	1.70	2.05	1.72
Average	2.76	1.93	2.33	1.90

Rate in bits per occupied voxel computed over the entire sequence.

the point-cloud frame at instant n, if we assume small motion in between the frames, we can use the past point-cloud frame as reference at instant $n-1$. Prediction and context can then be computed for all levels of the octree and the probabilities can be encoded and made available to the decoder side. We consider two methods to construct the reference point cloud to frame n. *Method I* utilizes the previous frame (at instant $n-1$) as is, without any compensation for motion, and all levels of the octree can be sequentially encoded. *Method II* progressively encodes levels. In it, assume we want to encode level k of the octree in frame n. We use levels 1 through k from frame $n-1$ and levels 1 through $k-1$ from frame n, to super-resolve frame n and estimate its kth level [11]. The super-resolved level k of the octree is used as a reference point cloud for encoding the kth level of the nth frame.

The probabilities (of a voxel being occupied) are computed at the encoder for each context. These probabilities are used to drive the arithmetic coder to encode the octree and sent to the decoder as side information. Essentially, for each context (distance from the voxel position to an occupied position in the previous frame), the model f is computed by counting how many voxels are occupied, then sending such a number to the decoder, i.e., we send one count per distance. In our notation, at level $\ell-1$, there are $N_{\ell-1}$ occupied voxels, yielding $K_\ell = 8N_{\ell-1}$ nodes for the next level. The K_ℓ nodes are divided into contexts, one for each distance $\delta_{\ell,k}$, each with $N_{\ell,k}$ nodes. We build the model and inform the probabilities to the decoder by simply conveying how many of the $N_{\ell,k}$ nodes are occupied (not leaves), which can be done with only $\lceil \log_2(N_{\ell,k}) \rceil$ bits since both encoder and decoder can deduce $N_{\ell,k}$. The efficient coding process for the side information yields a very small bit rate, accounting for typically less than 0.5% of the total. In our tests, the model f is computed for every frame based on the previous one, and the side-information rate is already included in the total bit rate.

While we explore contexts in the voxel domain, there are other techniques that use contexts within the octree-coded bitstream [8], [10] in methods that do not necessarily compete with our probability model. We believe the study presented in [10] is the state of the art in lossless compression of geometry in dynamic point clouds and a comparison against [8] is carried therein. We ran compression tests for a few point-cloud sequences, mostly from the MPEG test sets [12], and the results are shown in Table I. We have compared our two reference models (Methods I and II) against not using probability models (regular octree coding). We also included the results for the method in [10] to see how we would compare to the state of the art. The plots indicating the bit rate for each frame are shown in Fig. 4 for a few sequences.

Fig. 4. Results for encoding sequences comparing our Methods I and II against no-probability modeling (regular octree coding). Results for the method in [10] are also shown. From top to bottom, the sequences are "Man," "David," "Andrew," "Ricardo," and "Phil."

IV. DISCUSSION AND CONCLUSION

From the plots, we can see sizable performance gains using context modeling and arithmetic coding, provided there is a good reference to estimate the probabilities. From the exam-

ple, we can see that for many frames there is very little gain using *Method I*, which occurs when there is a strong motion activity so that the consecutive frames are not too similar. This problem is reduced by using a compensated reference model, as in *Method II*. On the other hand, when there is little or no motion, gains can be extreme, since the reference frame is an excellent prediction of the actual frame and distances to the reference voxels can be a reliable predictor of the actual voxel occupancy probability.

The robust reference model in *Method II* makes the final compression performance to be comparable with the state of the art. Despite the high-compression performance, we use the voxel-domain context for arithmetic coding, which does not prevent combining with the method in [10] which uses different contexts. We do believe a better compression algorithm will emerge when we combine the context modeling in [10] with voxel-distance-based probability models discussed here. Another possible improvement is to use motion compensation to move the points in the previous frame closer to the actual frame-point positions. There is still much room for improvement.

REFERENCES

[1] S. Orts-Escolano *et al.*, "Holoportation: Virtual 3D teleportation in real-time," in *Proc. 29th ACM User Interface Softw. Technol. Symp.*, Tokyo, Japan, Oct. 2016, pp. 741–754.

[2] C. Loop, C. Zhang, and Z. Zhang, "Real-time high-resolution sparse voxelization with application to image-based modeling," in *Proc. High-Perform. Graph. Conf.*, 2013, pp. 73–79.

[3] R. Schnabel and R. Klein, "Octree-based point-cloud compression," in *Proc. Symp. Point-Based Graph.*, Jul. 2006, pp. 111–121.

[4] D. Meagher, "Geometric modeling using octree encoding," *Comput. Graph. Image Process.*, vol. 19, no. 2, pp. 129–147, Jun. 1982.

[5] R. B. Rusu and S. Cousins, "3D is here: Point cloud library (PCL)," in *Proc. IEEE Int. Conf. Robot. Autom.*, Shanghai, China, May 2011, pp. 1–4. doi: 10.1109/ICRA.2011.5980567.

[6] C. Zhang, D. Florêncio, and C. Loop, "Point cloud attribute compression with graph transform," in *Proc. IEEE Int. Conf. Image Process.*, Sep. 2014, pp. 2066–2070.

[7] D. Thanou, P. A. Chou, and P. Frossard, "Graph-based motion estimation and compensation for dynamic 3D point cloud compression," in *Proc. IEEE Int. Conf. Image Process.*, Sep. 2015, pp. 3235–3239.

[8] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach, "Real-time compression of point cloud streams," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2012, pp. 778–785.

[9] R. de Queiroz and P. Chou, "Compression of 3D point clouds using a region-adaptive hierarchical transform," *IEEE Trans. Image Process.*, vol. 25, no. 8, pp. 3497–3956, Aug. 2016.

[10] D. C. Garcia and R. L. de Queiroz, "Context-based octree coding for point-cloud video," in *Proc. IEEE Int. Conf. Image Process.*, Beijing, China, Sep. 2017, pp. 1412–1416.

[11] D. C. Garcia, T. A. Fonseca, and R. L. de Queiroz, "Example-based super-resolution for point cloud video," arXiv:1803.06466, Mar. 17, 2018.

[12] C. Loop, Q. Cai, S.O. Escolano, and P. A. Chou, "Microsoft voxelized upper bodies - A voxelized point cloud dataset," ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG), Input Doc. M38673/M72012, May 2016.