Compression of 3D Point Clouds Using a Region-Adaptive Hierarchical Transform

Ricardo L. de Queiroz, *Senior Member, IEEE*, and Philip A. Chou, *Fellow, IEEE*

***Abstract*— In free-viewpoint video, there is a recent trend to represent scene objects as solids rather than using multiple depth maps. Point clouds have been used in computer graphics for a long time, and with the recent possibility of real-time capturing and rendering, point clouds have been favored over meshes in order to save computation. Each point in the cloud is associated with its 3D position and its color. We devise a method to compress the colors in point clouds, which is based on a hierarchical transform and arithmetic coding. The transform is a hierarchical sub-band transform that resembles an adaptive variation of a Haar wavelet. The arithmetic encoding of the coefficients assumes Laplace distributions, one per sub-band. The Laplace parameter for each distribution is transmitted to the decoder using a custom method. The geometry of the point cloud is encoded using the well-established octtree scanning. Results show that the proposed solution performs comparably with the current state-of-the-art, while being much more computationally efficient. We believe this paper represents the state of the art in intra-frame compression of point clouds for real-time 3D video.**

***Index Terms*— Point cloud compression, 3D immersive video, free-viewpoint video, RAHT, real-time point cloud transmission.**

I. Introduction

With dynamic 3D data, 3D representations of active people can be captured and conveyed in real time to remote locations, enabling free viewpoint viewing and rich collaboration as if all parties were co-located [1]. An illustration of free-viewpoint viewing is given in Fig. 1. Dynamic 3D data capture can be implemented using multiple cameras (infrared and regular) while visualization can use special glasses to render the subject within a synthetic or real scene. The processing for capture and display can be done in real time using powerful graphics processing units (GPUs) [2]. We are seeing the dawn of a new era when real time transmission in 3D is finally becoming a reality.

The compression of data for 3D transmission has been pursued mostly in the context of compression of meshes [3]–[12]. However, for real-time processing, the representation of scenes as point clouds seems to be more computationally efficient than meshes [2]. Previous approaches to compressing the geometry of point clouds include [13]–[15]. In [16] there is an approach to compress both color and geometry, while [17] presents a method to compress colors in the cloud, while leaving the geometry to be compressed using the known octtree method. We believe [17] and [18] represent the state-of-the-art in point cloud color compression, with the former focusing on intra-frame compression and the latter extending the former work to inter-frame compression. Both use the same codec, applied either to the colors directly or to their prediction residuals, based on an orthogonal graph transform and arithmetic coding of carefully modeled coefficients. The graph transform is a natural choice for the spatial transform of the color signal due to the irregular domain of definition of the signal. Unfortunately, the graph transform requires repeated eigen-decompositions of many and/or large graph Laplacians, rendering the approach infeasible for real-time processing.

Other approaches to transform signals over irregular domains of definition have been either to use shape-adaptive transforms, e.g., [19], or to pad the signal out to a regular domain and then use an ordinary block transform, e.g., [20]. Unfortunately the former sacrifices orthogonality, while the latter sacrifices critical sampling, both of which become extreme in the case of 3D point clouds, as the region of support becomes essentially only a 2D manifold within a 3D space.

In our work, we develop a region-adaptive orthogonal transform suitable for compression of color signals on 3D point clouds that not only compares well, in terms of rate-distortion performance, to the state-of-the-art systems that use the graph transform, but also is far superior to those systems in terms of computational complexity, easily enabling color compression of 3D point clouds in real time. The transform is hierarchical, resembling an adaptive variation of a Haar wavelet. We couple the transform with a novel feedforward approach to entropy coding the quantized transform coefficients. Geometry compression is discussed for completeness and we use the well established octtree method [13]–[17].

In this paper, we focus on intra-frame compression. Application to inter-frame compression is left to subsequent work. Also, we are only concerned with a real time transmission system. Each cloud has about 300K-400K points in a voxel grid and each frame has to be displayed at a rate of at least 30 frames per second (fps), yielding less than 30 ms to fully process each frame. Hence, we propose a low-complexity compression system suitable for real-time implementation, in contrast to the graph-transform method, which can be fairly complex.

Section II describes the geometry aspects of 3D data and how we encode the 3D position information. Section III

Manuscript received August 24, 2015; revised November 23, 2015; accepted December 5, 2015. Date of publication June 1, 2016; date of current version June 28, 2016. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Rahul Vanam.

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

P. A. Chou is with Microsoft Research, Redmond, WA 98052-6399 USA (e-mail: pachou@microsoft.com).

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

Digital Object Identifier 10.1109/TIP.2016.2575005

Fig. 1. Random viewpoints of a 3D point cloud frame.

describes the transform used in this work and Section IV presents the method to encode the resulting coefficients. Experimental results are presented and discussed in Sec. V and, at last, Section VI contains the conclusions of the present work.

II. Geometry Encoding

In a point cloud representing a 3D object, each volumetric element or *voxel* is associated with geometrical or appearance attributes. Our interest lies on a very simplified model which is being used for real-time applications. In it, voxels are occupied or not. An unoccupied voxel is transparent and devoid of other properties. Occupied voxels, in this simpler model, are associated with geometry (a position in the space) and a color. Techniques for capturing and rendering such models can be found elsewhere [2]. We are only interested in compressing the point cloud representation, which comprises of a list voxels $\{v_i\}$, each being described by its geometry (location in space) and color (in RGB or YUV color spaces), i.e.,

$$v_i = [x_i, y_i, z_i, Y_i, U_i, V_i]. \tag{1}$$

For a number of practical reasons, the geometry $\{[x_i, y_i, z_i]\}$ is encoded using the octtree scanning of the voxels, which has been shown to be very efficient [2], [14], [16].

A. Octtree Scanning

The octtree is the 3D extension of a 2D quad-tree. Assume, for convenience, that the volume to be represented is a cube of dimensions $W \times W \times W$ meters. In the first level of the octtree, the signal is partitioned into 8 smaller cubes of dimensions $W/2 \times W/2 \times W/2$, as depicted in Fig. 2. In a second level, each of the cubes is further partitioned in exactly the same way, each generating 8 cubes of dimensions $W/4 \times W/4 \times W/4$. If all cubes are split we would obtain 64 cubes of reduced dimensions. The process can be repeated for L levels yielding 2^{3L} voxels of dimensions $2^{-L}W \times 2^{-L}W \times 2^{-L}W$.

As for encoding the geometry, one could signal with a binary symbol if the voxel is occupied or not. However, since we are dealing with signals wherein less than 1% of total voxels are occupied, we use the octtree scanning to take advantage of the large unoccupied areas. In the first level, the entire space is segmented into 8 cubes. For each one we check if they are occupied or not, i.e., if any of the $\{x_i, y_i, z_i\}$ in the list of voxels belong to the cube. If so, we mark this cube with a binary symbol 1, else we mark it with a 0. That will take 8 bits. We, then, process the next level and only the cubes which are marked 1 are subject to further splitting, as illustrated in Fig. 3. A node marked 0 is a leaf in the octtree. The process is continued for L levels, only partitioning cubes which are occupied, resulting in large unoccupied areas represented by the tree leaves, and occupied voxels of dimensions $2^{-L}W \times 2^{-L}W \times 2^{-L}W$.

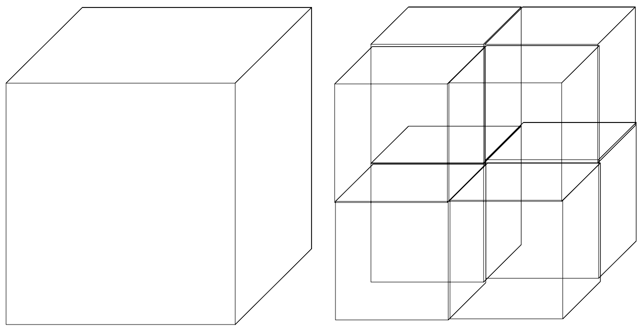


Fig. 2. In a particular level, a unit cube is divided into 8 sub-cubes, by splitting each dimension into two halves. Each sub-cube is further subdivided in the same manner. After L levels, starting with a cube of sides W we reach individual voxels whose dimensions are $2^{-L}W$.

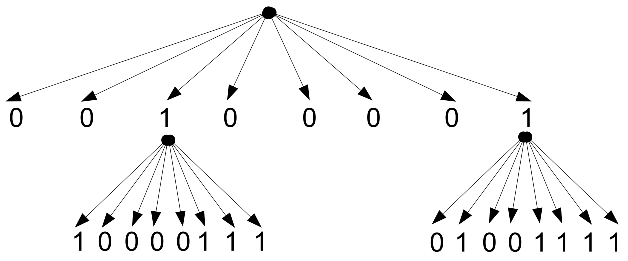


Fig. 3. Spanning the space, traversing the octtree. Unoccupied sub-cubes are made leaves of the tree and the occupied ones are further subdivided.

B. Encoding Rate

With just one level, there are only 8 sub-blocks, which are represented with 8 bits. If n_1 bits of the root are 1's, a second level decomposition in the octtree representation would demand another $8n_1$ bits. If in this second level there are n_2 bits 1, then the next level would demand $8n_2$ bits and so forth. Hence, the number of bits used to represent the octtree is:

$$B = 8 \sum_{k=0}^{L-1} n_k, \tag{2}$$

where we assume $n_0 = 1$.

For an L-level decomposition, we have a representation for a space of 2^{3L} voxels, of which only N_v voxels are occupied. We are interested in the bit-rate $R_g = B/N_v$, in bits-per-voxel, necessary to encode the geometry, i.e., to indicate where the occupied voxels are.

We are mostly interested in representing the surface of solid 3D objects. If we consider small sub-blocks, the intersection of the object surface with the block would be approximately planar (flat). We can, for example, estimate R_g if we assume the voxels lie in a flat surface crossing the entire voxel space. If one slices a unit cube into two pieces through a planar cut, the area of the cross section would lie somewhere between

0 and $\sqrt{2}$, with a cross section of area 1 being the most frequent. If the number of levels is L, the space is subdivided into 2^{3L} voxels. Assume also that the voxel space and the voxels are cubic, for simplicity. A slice of the voxel space parallel to the external faces should encompass 2^{2L} voxels and we assume our surface to have a number of voxels proportional to this number, i.e., $N_v = \beta 2^{2L}$.

In the first level of the octtree we use 8 bits. If a fraction α of the child nodes (sub-blocks) is occupied, 8α blocks are to be further subdivided. Still using the planar object assumption, in average α can be assumed constant throughout all blocks and levels. Hence, the number of bits used would be $8 + 8\alpha$ $(8 + 8\alpha(8 + 8\alpha \ldots)))$ for L levels, which amounts to $8 + 8(8\alpha) + 8(8\alpha)^2 + 8(8\alpha)^3 + \ldots$. Hence, our estimate is

$$R_g = \frac{B}{N_v} \approx \frac{8}{2^{2L}\beta} \sum_{k=0}^{L-1} (8\alpha)^k. \tag{3}$$

As L increases,

$$R_g \to \frac{2^{3+L}\alpha^L}{\beta(8\alpha - 1)} \tag{4}$$

and if we assume typical values of $\beta = 1$ and $\alpha = 1/2$, we get

$$R_g \approx \frac{8}{3}\text{bpv}. \tag{5}$$

This estimate for the geometry bit-rate has been shown to be very accurate in practice, so that the rule-of-thumb for geometry encoding is to expect to spend around 2.5 bits per occupied voxel using octtree scanning, which is confirmed by the experiments of the authors of [2] and the results in [15].

III. Region-Adaptive Hierarchical Transform

We derive a transform inspired by the idea of using the colors associated with a node in a lower level of the octtree to predict the colors of the nodes in the next level. We follow the octtree scan backwards, from voxels to the entire space, at each step recombining voxels into larger ones until reaching the root. At each decomposition, instead of grouping eight voxels at a time, we do it in three steps along each dimension, (e.g., along x, then y then z), so that we take $3L$ levels to traverse the tree backwards.

Let $g_{l,x,y,z}$ be the (scaled) average voxel color at level l, for x, y, z integers. $g_{l,x,y,z}$ is obtained by grouping (i.e. taking a linear combination of) $g_{l+1,2x,y,z}$ and $g_{l+1,2x+1,y,z}$, where the grouping along the first dimension is an example. We only process occupied voxels. If one of the voxels in the pair is unoccupied, the other one is promoted to the next level, unprocessed, i.e., $g_{l-1,x,y,z} = g_{l,2x,y,z}$ if the latter is the occupied voxel of the pair. The grouping process is repeated until getting to the root. Note that the grouping process generates voxels at lower levels that are the result of grouping different numbers of voxels along the way. The number of voxels grouped to generate voxel $g_{l,x,y,z}$ is the *weight* $w_{l,x,y,z}$ of that voxel.

So far, the notion of grouping neighboring voxels has been vaguely stated and we have not explained the transform coefficients to be encoded. For that, at every grouping of two voxels, say $g_{l,2x,y,z}$ and $g_{l,2x+1,y,z}$, with their respective weights, $w_{l,2x,y,z}$ and $w_{l,2x+1,y,z}$, we apply the following transform:

$$\begin{bmatrix} g_{l-1,x,y,z} \\ h_{l-1,x,y,z} \end{bmatrix} = \mathbf{T}_{w_1\ w_2} \begin{bmatrix} g_{l,2x,y,z} \\ g_{l,2x+1,y,z} \end{bmatrix}, \tag{6}$$

where $w_1 = w_{l,2x,y,z}$ and $w_2 = w_{l,2x+1,y,z}$ and

$$\mathbf{T}_{w_1 w_2} = \frac{1}{\sqrt{w_1 + w_2}} \begin{bmatrix} \sqrt{w_1} & \sqrt{w_2} \\ -\sqrt{w_2} & \sqrt{w_1} \end{bmatrix} \tag{7}$$

is the actual transform. Note that the transform matrix changes at all times, adapting to the weights, i.e., adapting to the number of leaf voxels that each $g_{l,x,y,z}$ actually represents. The quantities $g_{l,x,y,z}$ are used to group and compose further voxels at a lower level. $h_{l,x,y,z}$ are the actual *high-pass* coefficients generated by the transform to be encoded and transmitted. Furthermore, weights accumulate for the level above. In the above example,

$$w_{l-1,x,y,z} = w_{l,2x,y,z} + w_{l,2x+1,y,z}. \tag{8}$$

In the last stage, the tree root, the remaining two voxels $g_{1,0,0,0}$ and $g_{1,0,0,1}$ are transformed into the final two coefficients as:

$$\begin{bmatrix} g_{DC} \\ h_{0,0,0,0} \end{bmatrix} = \mathbf{T}_{w_{1000} w_{1001}} \begin{bmatrix} g_{1,0,0,0} \\ g_{1,0,0,1} \end{bmatrix}, \tag{9}$$

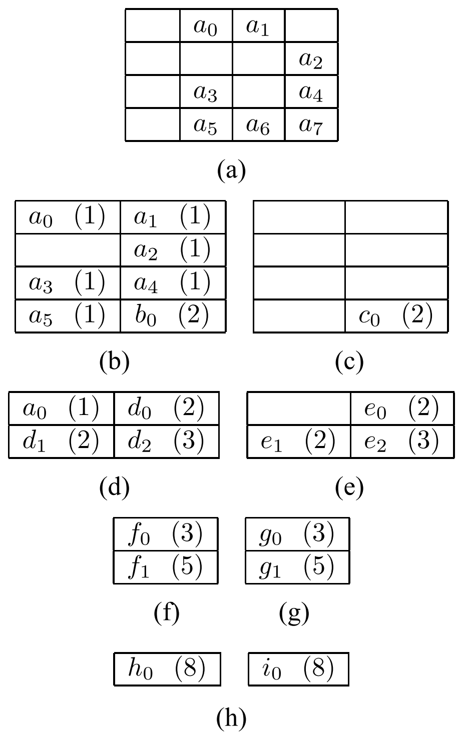
where $g_{DC} = g_{0,0,0,0}$. Note that since $\mathbf{T}_{w_1 w_2}$ is orthonormal at every level, the entire transform, as a composition of orthonormal transforms, is orthonormal. This is an extremely important property for compression, as the norm of the quantization error in the transform domain remains the same in the signal domain.

Formally, the transform can be described recursively as follows. Let $s_1 = (s_{11}, \ldots, s_{1w_1})$ and $s_2 = (s_{21}, \ldots, s_{2w_2})$ be signal vectors defined over non-intersecting domains (i.e., regions) $\mathcal{R}_1$ and $\mathcal{R}_2$, respectively, where $w_1 = |\mathcal{R}_1|$ and $w_2 = |\mathcal{R}_2|$, and let their transforms be $t_1 = (t_{11}, \ldots, t_{1w_1})$ and $t_2 = (t_{21}, \ldots, t_{2w_2})$, respectively. Let $s = (s_{11}, \ldots, s_{1w_1}, s_{21}, \ldots, s_{2w_2})$ be the concatenated signal vector defined over $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$. Then the $(w_1 + w_2)$-point transform of s is

$$t = (at_{11} + bt_{21}, t_{12}, \ldots, t_{1w_1}, -bt_{11} + at_{21}, t_{22}, \ldots, t_{2w_2}), \tag{10}$$

where $a = \sqrt{w_1/(w_1 + w_2)}$ and $b = \sqrt{w_2/(w_1 + w_2)}$. That is, the matrix (7) is applied to the first ("DC") components of t_1 and t_2 (namely t_{11} and t_{21}) to obtain the first ("DC") and $(w_1 + 1)$th components of t, while the other components remain unchanged. As the base case, the transform of a scalar is the scalar itself. Note that the transform can be performed "in place." Also note that the domain of definition is a discrete abstract set and can be embedded in any dimension.

We use a 2D example to further illustrate the process. Figure 4(a) depicts a 2D map of voxels (or pixels) labeled $\{a_i\}$ that one wants to group and transform. The unoccupied voxels are left empty. In the first level, we group horizontal pairs, from which we can see that the pair (a_6, a_7) is transformed using $\mathbf{T}_{11}$ yielding another voxel b_0 and a high-pass coefficient c_0. Hence, one can say that the voxel map in Fig. 4(a)



a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7
(a)
a_0 (1) a_1 (1) a_2 (1) a_3 (1) a_4 (1) a_5 (1) b_0 (2)
(b)
c_0 (2)
(c)
a_0 (1) d_0 (2) d_1 (2) d_2 (3)
(d)
e_0 (2) e_1 (2) e_2 (3)
(e)
f_0 (3) f_1 (5)
(f)
g_0 (3) g_1 (5)
(g)
h_0 (8)
i_0 (8)
(h)

$$\begin{bmatrix} b_0 \\ c_0 \end{bmatrix} = \mathbf{T}_{11} \begin{bmatrix} a_6 \\ a_7 \end{bmatrix} \qquad \begin{bmatrix} d_0 \\ e_0 \end{bmatrix} = \mathbf{T}_{11} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

$$\begin{bmatrix} d_1 \\ e_1 \end{bmatrix} = \mathbf{T}_{11} \begin{bmatrix} a_3 \\ a_5 \end{bmatrix} \qquad \begin{bmatrix} d_2 \\ e_2 \end{bmatrix} = \mathbf{T}_{12} \begin{bmatrix} a_4 \\ b_0 \end{bmatrix}$$

$$\begin{bmatrix} f_0 \\ g_0 \end{bmatrix} = \mathbf{T}_{12} \begin{bmatrix} a_0 \\ d_0 \end{bmatrix} \qquad \begin{bmatrix} f_1 \\ g_1 \end{bmatrix} = \mathbf{T}_{23} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$$

$$\begin{bmatrix} h_0 \\ i_0 \end{bmatrix} = \mathbf{T}_{35} \begin{bmatrix} f_0 \\ f_1 \end{bmatrix}$$

(i)

Fig. 4. Example in 2D to illustrate the grouping to form the hierarchical transform: (a) voxel map to be transformed into (b) low-pass and (c) high-pass maps by grouping horizontal neighbors. The low-pass map in (b) is further subdivided into (d) low-pass and (e) high-pass by grouping this time the neighbors in the vertical direction. The process is repeated from (d) into (f) and (g) and from (f) into the two final coefficients in (h). The transforms are indicated in (i). The voxel weights are indicated along with the labels.

generated a low-pass map in Fig. 4(b) and a high-pass map in Fig. 4(c). In those, the number in parentheses is the weight of a given voxel. The map in Fig. 4(b) is further grouped along the vertical direction yielding the new low- and high-pass maps in Fig. 4(d) and (e), respectively, which contain the low-pass voxels $\{d_i\}$ and the high-pass coefficients $\{e_i\}$. The map in Fig. 4(d) is further subdivided into the low- and high-pass maps in Fig. 4(f) and (g), respectively. In the final step the two elements in Fig. 4(f) are transformed into two coefficients in Fig. 4(h). h_0 corresponds to g_{DC} and i_0 is a high-pass coefficient. Note that the coefficients to be encoded and transmitted in this example are $c_0, e_0, e_1, e_2, g_0, g_1, i_0$ and h_0, which are the same, in number, as voxels in Fig. 4(a). The transform matrices at each stage are in Fig. 4(i).

It can be seen that the process reduces to the Haar transform when the weights are uniform, which is the case when the region is regular, e.g. fully occupied. Thus we view the process as a *region-adaptive Haar* (or *hierarchical*) *transform* (RAHT). We define a sub-band as the collection of all high-pass coefficients that are associated with the same weights, i.e., h_{l_1,x_1,y_1,z_1} is in the same sub-band as h_{l_2,x_2,y_2,z_2} if and only if $w_{l_1,x_1,y_1,z_1} = w_{l_2,x_2,y_2,z_2}$. The DC coefficient is treated as forming its own sub-band.

There are N_v occupied voxels. After transformation, there are S sub-bands, each sub-band with N_i transformed coefficients. Since the transform is non-expansive, the number of transformed coefficients and occupied voxels is the same, i.e., $\sum_{m=0}^{S-1} N_m = N_v$. If $H(\,)$ is our hierarchical transform, which depends on the geometry of all N_v voxels $\{v_k\}$, then

$$\begin{aligned} \{f_Y(m,n)\} &= H(\{Y_i\}) \\ \{f_U(m,n)\} &= H(\{U_i\}) \\ \{f_V(m,n)\} &= H(\{V_i\}) \end{aligned} \tag{11}$$

for $0 \le m < S$ being the sub-band index and $0 \le n < N_m$ being the coefficient index within the mth sub-band.

It is apparent that only modest computation is needed for the RAHT. As a result, it has been successfully programmed to process a frame under 33 ms (30 fps real-time system requirement) using common graphic processors.

IV. Coefficient Encoding

We quantize the transform coefficients using a uniform scalar quantizer, and then entropy code each quantized coefficient using an arithmetic coder (AC) [21]. The AC is driven by a Laplacian distribution whose parameter, unique to each sub-band, is encoded and transmitted from the encoder to the decoder.

A. Arithmetic Encoding of a Laplacian Distributed Coefficient

We assume that each transform coefficient X within a given sub-band has a Laplacian density with parameter b,

$$p(x) = \frac{1}{2b} e^{-\frac{|x|}{b}}, \tag{12}$$

i.e., variance $\sigma^2 = 2\ b^2$, and that it is uniformly scalar quantized with stepsize Q,

$$k = \mathrm{round}(X/Q). \tag{13}$$

The probability that X lies in the kth quantization bin is thus

$$\begin{aligned} p_k &\overset{\Delta}{=} P((k-1/2)Q \le X < (k+1/2)Q) && (14) \\ &= \int_{(k-\frac{1}{2})Q}^{(k+\frac{1}{2})Q} \frac{1}{2b} e^{-\frac{|x|}{b}} dx && (15) \\ &= \begin{cases} \frac{1}{2} e^{-\frac{|k|Q}{b}} \left(e^{\frac{Q}{2b}} - e^{-\frac{Q}{2b}} \right) & k \ne 0 \\ 1 - e^{-\frac{Q}{2b}} & k = 0, \end{cases} && (16) \end{aligned}$$

so that the fractional rate (in natural information units, nats) used by the AC to encode the quantized coefficient if it falls into the kth quantization bin is precisely [21]

$$r(k,b) = -\ln p_k = \begin{cases} \frac{|k|Q}{b} - \ln\left(\sinh(\frac{Q}{2b})\right) & k \neq 0 \\ -\ln\left(1 - e^{-\frac{Q}{2b}}\right) & k = 0. \end{cases} \quad (17)$$

When $Q \ll 2b$, this is well-approximated for all k by

$$r(k,b) \approx \tilde{r}(k,b) = \frac{|k|Q}{b} - \ln\left(\frac{Q}{2b}\right). \quad (18)$$

This is just a probability model for the AC and any parameter b can be used, as long as both the encoder and decoder agree on it. In order to find the best parameter choice, summing (18) over all symbols k_n in a sub-band, taking the derivative with respect to b, and equating to zero,

$$\frac{d}{db}\sum_{n=1}^{N}\tilde{r}(k_n,b) = \sum_{n=1}^{N}\left(-\frac{|k_n|Q}{b^2} + \frac{1}{b}\right) = 0, \quad (19)$$

we find that the value of b that minimizes the number of bits needed to encode all N symbols in the sub-band is

$$b^* = \frac{1}{N}\sum_{n=1}^{N}|k_n|Q. \quad (20)$$

B. Conveying the Sub-Band Parameter to the Decoder

At this point let us include the sub-band index in our notation, e.g., b_m^* for the optimal parameter of the mth sub-band.

We explicitly convey the optimal parameters $\{b_m^*\}$ to the decoder so that they can be used in the arithmetic coder. To convey the parameters, we need to quantize and encode them. Since quantization will introduce some errors in the parameters, the total number of bits used for the coefficients will slightly increase. To determine the optimal stepsize for the parameters, we need to trade off this increase with the number of bits needed to encode the parameters themselves.

To be precise, suppose we quantize the parameters in the log domain, $\phi_m^* = \ln b_m^*$, using a uniform scalar quantizer, and then entropy code the resulting sequence of symbols. For a quantized value $\phi = \phi_m^* + \epsilon$, the total number of nats needed to code the coefficients in sub-band m rises from its minimum $\sum_{n=1}^{N_m}\tilde{r}(k_{mn}, b_m^*)$ to

$$f(\phi) = \sum_{n=1}^{N_m}\tilde{r}(k_{mn}, e^{\phi}) \approx \sum_{n=1}^{N_m}\tilde{r}(k_{mn}, b_m^*) + \frac{N_m}{2}\epsilon^2, \quad (21)$$

which is the second-order Taylor expansion of $f(\phi)$ around ϕ_m^*. Note that N_m, which is the number of coefficients in sub-band m, is the second derivative of $f(\phi)$ at $\phi = \phi_m^*$:

$$\frac{d^2 f}{d\phi^2}(\phi_m^*) = \frac{d^2}{d\phi^2}\sum_{n=1}^{N_m}\left(\frac{|k_n|Q}{e^{\phi}} - \ln\left(\frac{Q}{2e^{\phi}}\right)\right)\Bigg|_{\phi=\phi_m^*} \quad (22)$$

$$= \sum_{n=1}^{N_m}\frac{|k_n|Q}{e^{\phi_m^*}} = \sum_{n=1}^{N_m}\frac{|k_n|Q}{b_m^*} = N_m. \quad (23)$$

Thus, $N_m\epsilon^2/2$ is the increase in the number of nats needed by the arithmetic coder to code the coefficients in sub-band m if the quantized log parameter $\phi_m^* + \epsilon$ is used instead of the optimal log parameter ϕ_m^*. Importantly, this increase is independent of the parameter, which allows the optimal stepsize of the quantizer to be independent of the parameter itself. To be specific, if δ_m is the stepsize of the quantizer, then according to high-rate quantizer theory [22] the expected squared error of the quantizer is approximately $\delta_m^2/12$, while its expected rate is approximately $h - \ln\delta_m$, where h is the differential entropy of the random variable to be quantized. Thus by adding $h - \ln\delta_m$ to the expected value of (21), we obtain the expected total number of nats for sub-band m,

$$h - \ln\delta_m + \sum_{n=1}^{N_m}\tilde{r}(k_{mn}, b_m^*) + \frac{N_m}{2}\frac{\delta_m^2}{12}, \quad (24)$$

which is minimized (taking the derivative with respect to δ_m and equating to zero) by

$$\delta_m = \frac{C}{\sqrt{N_m}} \quad (25)$$

for some constant C, which we find empirically. This equation tells us to use a quantizer stepsize that is inversely proportional to the square root of the number of coefficients in the sub-band. In our experiments, rather than quantize ϕ_m^*, we quantize b_m^* directly, which we find as effective in practice. We use a run-length Golomb-Rice (RLGR) coder [23] as the entropy code for the sequence of quantized parameters.

C. Encoding Coefficients

As seen in Sec. II, the geometry (the location of each occupied voxel) is separately encoded and conveyed to the decoder using octtrees. The construction of the hierarchical decomposition depends solely on the geometry, even though the transform is eventually applied to the color of the voxels. Hence, both encoder and decoder can exactly know how many sub-bands there are (S) and how many coefficients are there in each sub-band ($\{N_m\}$). The sub-band parameters $\{b_m^*\}$ are dependent on the color data and need to be conveyed to the decoder. The transformed coefficients $f_Y(m,n)$, $f_U(m,n)$, $f_V(m,n)$ are then uniformly scalar quantized and sent to the decoder using arithmetic coding. The compression steps are:

- Encode and send the geometry using octtree scanning.
- Encode each color component using the region-adaptive hierarchical transform, uniform quantization, and entropy coding.

To encode each color component, using luminance as an example, we take the following steps:

1) Transform the signal using the transform H as in (11): $\{f_Y(m,n)\} = H(\{Y_i\})$.
2) Quantize the transformed coefficients $f_Y(m,n)$ using a uniform quantizer with stepsize Q as in (13): $k_{mn} = \text{round}(f_Y(m,n)/Q)$.
3) Compute the optimal sub-band parameters b_m^* using (20): $b_m^* = (1/N_m)\sum_n |k_{mn}|Q$.

4) Quantize the sub-band parameters as $\gamma_m = \text{round}\left(b_m^* \sqrt{N_m}/C\right)$. The first (highest pass) sub-band has a large number of coefficients and a small parameter. The last two (lowest pass) sub-bands have only one coefficient each, which are very large. As exceptions, the optimal parameter of the first sub-band is encoded with a 24-bit floating point representation and the coefficients of the last two sub-bands are rounded and encoded using a fixed 16-bit integer representation. The overhead is insignificant in a cloud with hundreds of thousands of occupied voxels.
5) Encode and send the set of integers $\{\gamma_m\}$ using RLGR encoding.
6) Reconstruct the parameters of the sub-bands as $\hat{b}_m = \gamma_m C/\sqrt{N_m}$.
7) Encode and send the set of integers k_{mn} using an AC with a probability model based on a Laplace distribution with parameter $\hat{b}_m$.

The decoder operates in the reverse manner. It first decodes the geometry, which is the same for all color components. Then it decodes all $\hat{b}_m$ for all sub-bands for the luminance, followed by all k_{mn}, from which it can reconstruct an approximation of the coefficients as $\hat{f}_Y(m,n) = k_{mn}Q$. Finally it performs an inverse transform to reconstruct all $\{\hat{Y}_i\}$. Note that $\hat{Y}_i \neq Y_i$ since $\hat{f}_Y(m,n) \neq f_Y(m,n)$. The process is repeated for the two chrominance components.

V. Simulation Results

We have carried out many tests using frames extracted from sequences of dynamic point cloud data sets. Frames in the data set were captured under different conditions. Four of our frames were captured using a real-time high resolution sparse voxelization algorithm [2]. The cameras are pointed at the front of the subject covering roughly a cubic meter of space. The captured videos are processed on a high-end graphics card using a sparse voxelization algorithm in real time. The sparse voxelization algorithm outputs an octtree and we set $L = 9$, which yields a 512^3 voxel space. We used frames "Andrew", "Ricardo", "Phil" and "Sarah", for which N_v are 286934, 207077, 325077, and 301336, respectively. We also included frame "Man", with 223617 occupied voxels, in the test set. The latter was captured using a non-real-time system [25] and converted to a mesh representation, which was voxelized for the present paper. Our test set is rendered in Fig 5, from random viewpoints. We also included point clouds from other databases, not necessarily suitable for real-time telepresence, but helpful to measure the performance of the proposed coder. One of them is the "Skier" from the ITI database[1] which has 229492 voxels. The other is a collection of objects from the 3DCOMET dataset [24] tagged as "r_sh_th_04" and referred to here as "Objects", which has 111920 voxels. Both "Skier" and "Objects" were available as meshes and were voxelized to $L = 9$.

In our experiments, we used $C = 20$ and varied Q from 10 to 40. We measured rate in bits per occupied voxel (or bpv) and distortion in peak signal-to-noise ratio (PSNR in dB), comparing the luminance component of the original and reconstructed frames. The geometry is encoded without loss.

We tested our proposed coder based on RAHT against the alternatives. To our knowledge, the state-of-the-art coder is the one based on a graph transform (GT) [17], in which

[1]http://vcl.iti.gr/reconstruction/

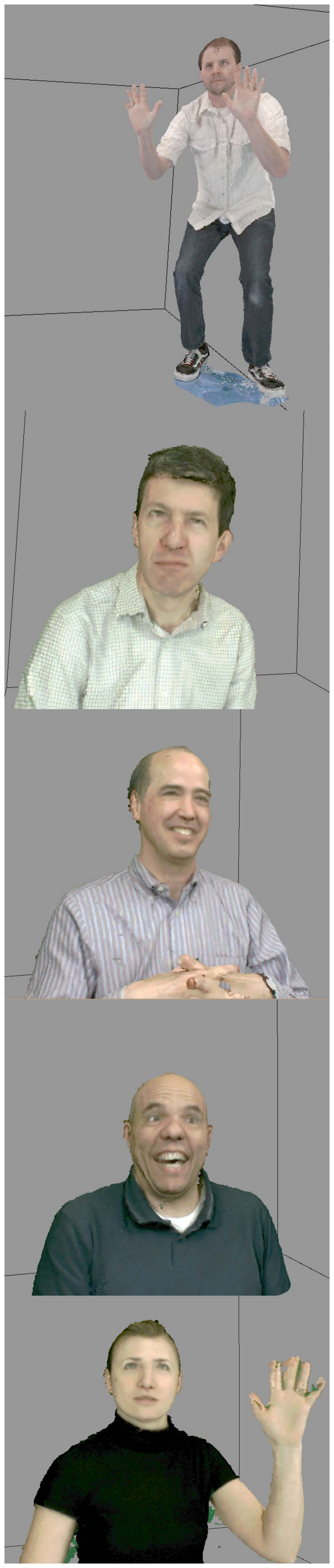


Fig. 5. Random renderings of our test set. From top to bottom: "Man", "Andrew", "Phil", "Ricardo" and "Sarah."

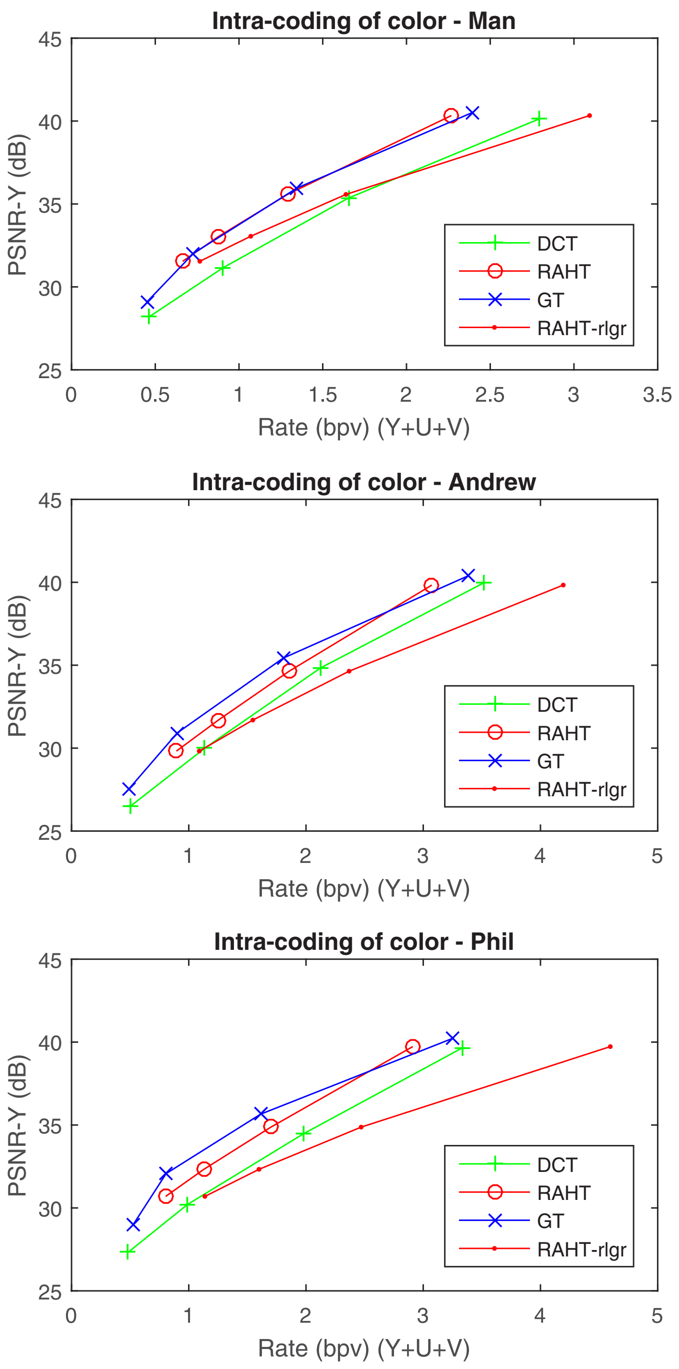


Fig. 6. RD curves for frames “Man”, “Andrew”, “Phil”.

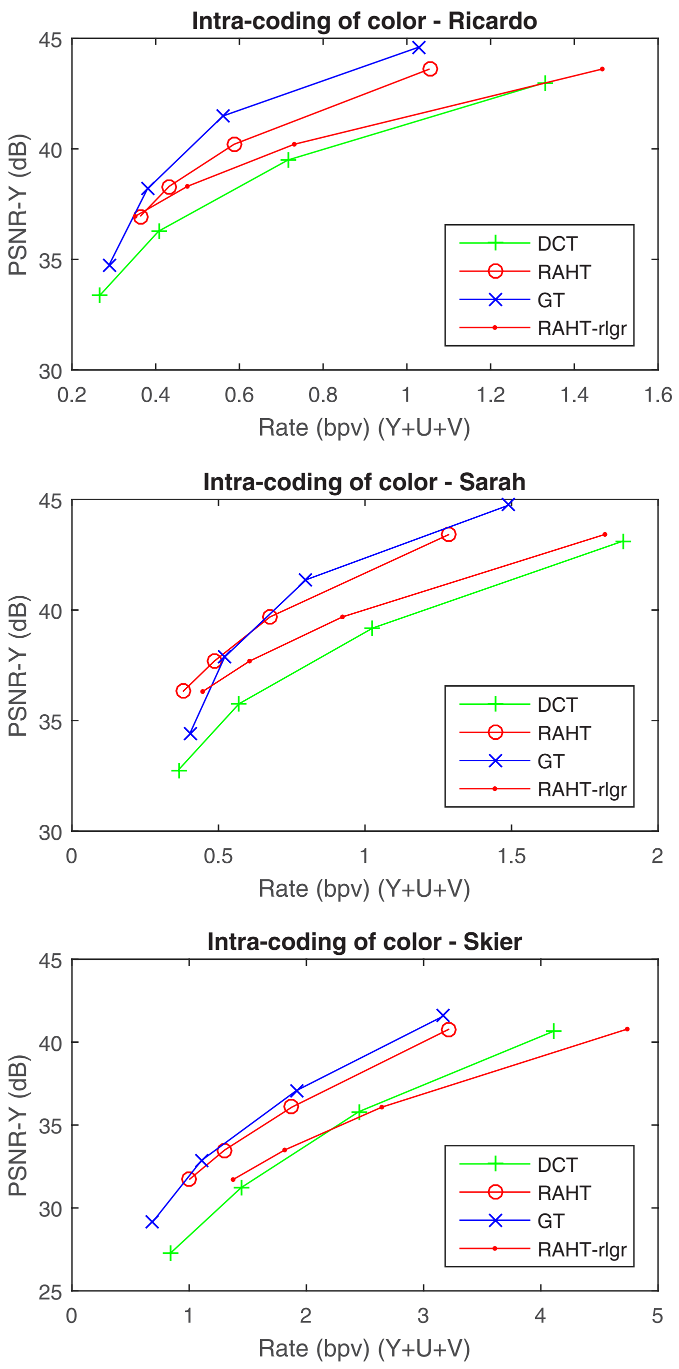


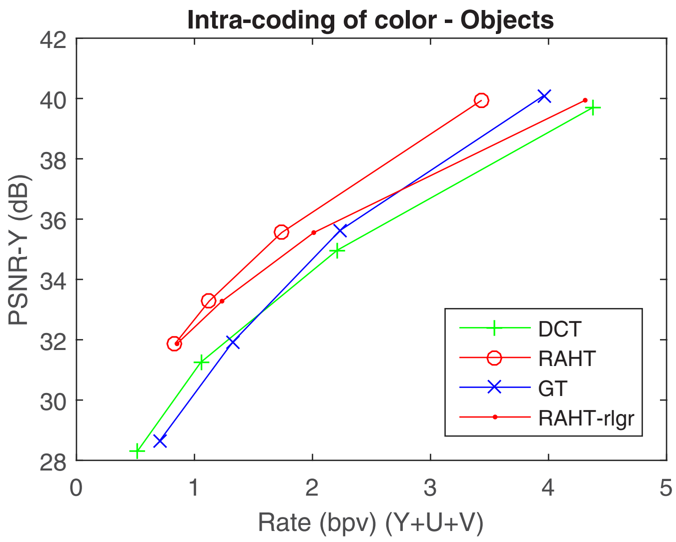
Fig. 7. RD curves for frames “Ricardo”, “Sarah” and “Skier”.

the frame is divided into cubes, for example into cubes of $8 \times 8 \times 8$ voxels. The occupied voxels in each cube are associated with their immediate neighbors in a graph and weights are attributed to the graph edges based on the distance in between voxels. The transform is found as the eigenvectors of the Laplacian matrix of the graph and the associated eigenvalues are used as estimates of the standard deviations used for the AC. Both the GT and RAHT coders use AC and the Laplace distribution. The GT-based coder derives the standard deviations while our RAHT-based coder explicitly transmits them.

A special case of the GT is when one connects the voxels into a line graph in which case the GT becomes exactly the discrete cosine transform (DCT) of the color values, sorted in some order within the cube.

A lower complexity variation of the RAHT-based coder that also might be of interest is to replace the AC in the RAHT with RLGR for each $\{f_{Y,U,V}(m,n)\}$. We refer to it as RAHT-RLGR.

Rate-distortion results are shown in Figs. 6, 7, and 8 for our data sets and four coders (RAHT, GT, DCT, and RAHT-RLGR). From the results, we can infer that the performance of the RAHT-based coder is comparable to the GT-based coder, slightly outperforming it in many tests. However, the GT is much more complex than the RAHT since it demands computing eigenvalues and eigenvectors of



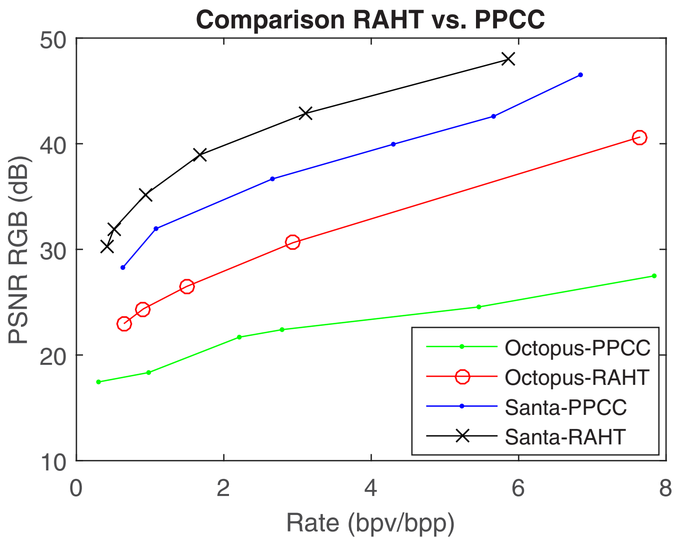


Fig. 8. RD curves. (a) Frame "Objects"; (b) comparison against PPCC for point clouds "Octopus" and "Santa".

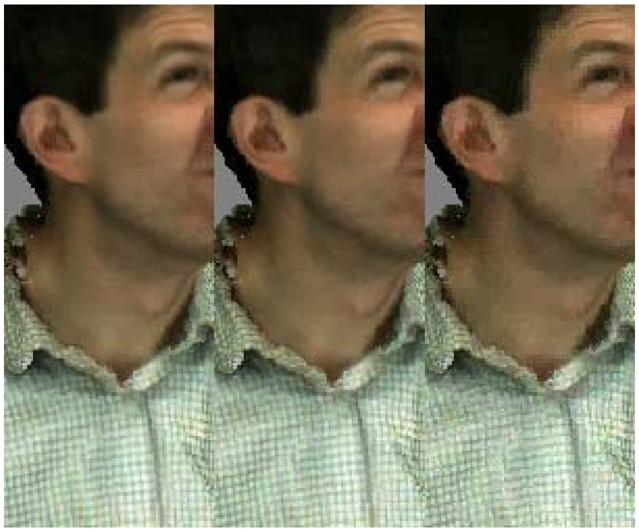


Fig. 9. Subjective comparison of RAHT performance. A zoom of a view of frame "Andrew" for 3 cases: (left) original; (center) after color compression to 2.5 bpv; (right) after color compression to 0.8 bpv.

large matrices. A single $8 \times 8 \times 8$ voxel cube may demand processing matrices of dimensions up to 512×512, and there are thousands of occupied cubes. That contrasts with the RAHT, whose complexity is nearly constant and proportional to N_v. Another advantage of the RAHT is that it can explore deeper levels of the octtree to remove data redundancies, in contrast to a relatively small size of the cubes in GT. The DCT approach has also a low complexity but its performance is inferior to the RAHT-based coder. Finally, the coder based on RAHT and RLGR is also a low-complexity option, even though it has shown a performance inferior to the coder based on RAHT and AC.

For completeness we also compare our RAHT-based coder to the Progressive Point Cloud Coder (PPCC) of Huang et al. [16]. They used a prediction pyramid and Lloyd-Max scalar quantization for color compression, whereas we use a critically sampled orthogonal transform and uniform scalar quantization. They reported color coding results for three data sets: "Octopus" (466K points), "Santa" (76K points), and "FemaleWB" (148K points). The data is no longer publicly available and we obtained the data sets directly from the PPCC authors. However, for the "FemaleWB" data set we received, the size was 122K, rather than the reported 148K, indicating a probable reprocessing. We voxelized the point clouds "Octopus" and "Santa" to $L = 17$ and $L = 12$ in order to ensure that no two points would coincide in a voxel. Thus, bits per (occupied) voxel (bpv) is equivalent to bits per point (bpp) in this case. Furthermore, in order to make our results compatible with those in [16] we computed the PSNR in RGB space. Results are shown in Fig. 8(b), where it can be seen that the RAHT-based coder significantly outperforms PPCC.

There is also the color coder in the Point Cloud Library [26] used in the 3DCOMET test set [24], which simply represents the colors with fewer bits. Such a method can have its PSNR theoretically derived using the $\Delta^2/12$ rule for computing the mean-squared error [22], so that for retaining k bits, the PSNR (in dB) is $10.75 + 6.02k$. Its RD curve would be so inferior to that of the RAHT-based coder that we did not include it in any of our plots.

Figure 9 shows an enlarged portion of the projection of frame "Andrew" for subjective evaluation. Three cases are presented: (a) without color compression; (b) color compression at 2.50 bpv and PSNR of 39.8 dB; (c) color compression at 0.85 bpv and PSNR of 31.7 dB.

VI. Conclusions

We have presented a new algorithm for the compression of the colors of 3D point clouds of still frames. We have developed a region-adaptive hierarchical transform, RAHT, that works its way up the octtree using an adaptive 2-point orthogonal transform to ensure the whole transform is non-expansive and orthogonal. The transform, although adaptive, can be easily computed in graphics cards or CPUs at a rate of 30 fps. The coefficients may be encoded using arithmetic coding driven by Laplacian distribution models, whose parameters are specific for each sub-band. The parameters are encoded with a custom coder and are transmitted in parallel to the decoder. The decoder decodes the parameters, and then configures its arithmetic decoders to decode the RAHT coefficients. The inverse RAHT computation is as complex as the forward one, making both excellent candidates for real time transmission of 3D point cloud data.

We conclude that the proposed RAHT-based color coder can be made to operate at a performance comparable to the GT-based one for a fraction of its computation cost. We also conclude, within our coder, the use of Golomb-Rice codes, although more GPU-friendly than AC, still needs improvement.

Future work may concentrate on different fronts. We need to better adapt RLGR to the RAHT coefficients. We also plan to use different transform sizes and to change them adaptively, i.e., to transform more than a pair of voxels at a time. Finally, we plan to use our approach to encode motion vectors and color prediction residuals for dynamic 3D data.

ACKNOWLEDGMENT

The authors would like to thank Dinei Florencio, Cha Zhang, Charles Loop, and Qin Cai for the use of their code (GT and DCT) and data ("Andrew", "Phil", "Ricardo", and "Sarah"), the Microsoft HCap group for the use of their data ("Man"), Jingliang Peng and Yuxue Fan for the use of their data ("Santa" and "Octopus") and the code to parse it, and Sergio Orts-Escolano for his help with the 3DCOMET dataset.

REFERENCES

[1] C. Zhang, Q. Cai, P. A. Chou, Z. Zhang, and R. Martin-Brualla, "Viewport: A distributed, immersive teleconferencing system with infrared dot pattern," *IEEE MultiMedia*, vol. 20, no. 1, pp. 17–27, Jan./Mar. 2013.

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

[4] O. Devillers and P.-M. Gandoin, "Geometric compression for interactive transmission," in *Proc. IEEE Visualizat.*, Oct. 2000, pp. 319–326.

[5] L. Váša and V. Skala, "Geometry-driven local neighbourhood based predictors for dynamic mesh compression," *Comput. Graph. Forum*, vol. 29, no. 6, pp. 1921–1933, 2010.

[6] H. Q. Nguyen, P. A. Chou, and Y. Chen, "Compression of human body sequences using graph wavelet filter banks," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, May 2014, pp. 6152–6156.

[7] S.-R. Han, T. Yamasaki, and K. Aizawa, "Time-varying mesh compression using an extended block matching algorithm," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 11, pp. 1506–1518, Nov. 2007.

[8] S. Gupta, K. Sengupta, and A. Kassim, "Registration and partitioning-based compression of 3-D dynamic data," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 11, pp. 1144–1155, Nov. 2003.

[9] X. Gu, S. J. Gortler, and H. Hoppe, "Geometry images," in *Proc. Annu. Conf. Comput. Graph. Interact. Techn.*, 2002, pp. 355–361.

[10] H. M. Briceño, P. V. Sander, L. McMillan, S. Gortler, and H. Hoppe, "Geometry videos: A new representation for 3d animations," in *Proc. ACM Symp. Comput. Animation*, 2003, pp. 136–146.

[11] H. Habe, Y. Katsura, and T. Matsuyama, "Skinoff: Representation and compression scheme for 3D video," in *Proc. Picture Coding Symp.*, 2004, pp. 301–306.

[12] J. Hou, L.-P. Chau, N. Magnenat-Thalmann, and Y. He, "Compressing 3-D human motions via keyframe-based geometry videos," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 1, pp. 51–62, Jan. 2015.

[13] T. Ochotta and D. Saupe, "Compression of point-based 3D models by shape-adaptive wavelet coding of multi-height fields," in *Proc. Eurograph. Conf. Point-Based Graph.*, 2004, pp. 103–112.

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

[15] 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.

[16] Y. Huang, J. Peng, C.-C. J. Kuo, and M. Gopi, "A generic scheme for progressive point cloud coding," *IEEE Trans. Vis. Comput. Graphics*, vol. 14, no. 2, pp. 440–453, Mar./Apr. 2008.

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

[18] 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.*, Quebec City, QC, Canada, Sep. 2015, pp. 3235–3239.

[19] G. Minami, Z. Xiong, A. Wang, and S. Mehrotra, "3-D wavelet coding of video with arbitrary regions of support," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 9, pp. 1063–1068, Sep. 2001.

[20] R. L. de Queiroz, "On data filling algorithms for MRC layers," in *Proc. IEEE Int. Conf. Image Process.*, Sep. 2000, pp. 586–589.

[21] A. Said, *Introduction to Arithmetic Coding—Theory and Practice* Imag. Syst. Lab., HP Lab., Palo Alto, CA, USA, Tech. Rep. HPL-2004-76, Apr. 2004.

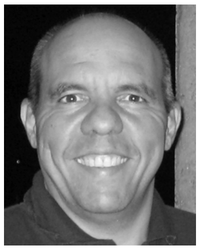
[22] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Hingham, MA, USA: Kluwer, 1992.

[23] H. S. Malvar, "Adaptive run-length/Golomb-Rice encoding of quantized generalized Gaussian sources with unknown statistics," in *Proc. Data Compress. Conf.*, Mar. 2006, pp. 23–32.

[24] J. Navarrete, V. Morell, M. Cazorla, D. Viejo, J. García-Rodríguez, and S. Orts-Escolano, "3DCOMET: 3D compression methods test dataset," *Robot. Autom. Syst.,* vol. 75, pp. 550–557, Jan. 2015.

[25] A. Collet *et al.*, "High-quality streamable free-viewpoint video," *ACM Trans. Graph.*, vol. 34, no. 4, 2015, Art. no. 69.

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



Ricardo L. de Queiroz (M'88–SM'99) received the Engineering degree from the Universidade de Brasília, Brazil, in 1987, the M.Sc. degree from the Universidade Estadual de Campinas, Brazil, in 1990, and the Ph.D. degree from The University of Texas at Arlington in 1994, all in electrical engineering. From 1990 to1991, he was with the DSP Research Group, Universidade de Brasília, as a Research Associate. He joined Xerox Corporation in 1994, where he was a member of the Research Staff until 2002. From 2000 to 2001, he was also an Adjunct Faculty Member with the Rochester Institute of Technology. He joined the Electrical Engineering Department, Universidade de Brasília, in 2003. In 2010, he became a Full Professor with the Computer Science Department, Universidade de Brasília. He was a Visiting Professor with the University of Washington, Seattle, in 2015. He has published over 160 articles in journals and conferences and contributed chapters to books as well. He also holds 46 issued patents. His research interests include image and video compression, multirate signal processing, and color imaging. He was an Elected Member of the IEEE Signal Processing Society's Multimedia Signal Processing and the Image, Video and Multidimensional Signal Processing Technical Committees. He is an Editor of the IEEE TRANSACTIONS ON IMAGE PROCESSING and was an Editor of the *EURASIP Journal on Image and Video Processing*, the IEEE SIGNAL PROCESSING LETTERS, and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY. He was appointed as an IEEE Signal Processing Society Distinguished Lecturer from 2011 to 2012. He has been actively involved with the Rochester chapter of the IEEE Signal Processing Society, where he served as a Chair and organized the Western New York Image Processing Workshop since its inception until 2001. He is now helping organizing IEEE SPS Chapters in Brazil and just founded the Brasilia IEEE SPS Chapter. He was the General Chair of ISCAS'2011, MMSP'2009, and SBrT'2012. He was also part of the organizing committee of ICIP'2002, ICIP'2012, ICIP'2014, and ICIP'2016. He is a member of the Brazilian Telecommunications Society.

Philip. A. Chou (M'81–SM'00–F'03) received the B.S.E. degree in electrical engineering and computer science from Princeton University, Princeton, NJ, in 1980, the M.S. degree in electrical engineering and computer science from the University of California, Berkeley, in 1983, and the Ph.D. degree in electrical engineering from Stanford University in 1988. From 1988 to 1990, he was a member of the Technical Staff with AT&T Bell Laboratories, Murray Hill, NJ. From 1990 to 1996, he was a member of the Research Staff with the Xerox Palo Alto Research Center, Palo Alto, CA. In 1997, he was a Manager of the Compression Group with VXtreme, an Internet video startup in Mountain View, CA, before it was acquired by Microsoft in 1997. Since 1998, he has been a Principal Researcher with Microsoft Research, Redmond, WA, managing the Communication and Collaboration Systems Research Group from 2004 to 2011. He served as a Consulting Associate Professor with Stanford University 1994–1995 and an Affiliate Associate Professor with the University of Washington 1998–2009, and has served as an Adjunct Professor with the Chinese University of Hong Kong since 2006. He has longstanding research interests in signal data compression, signal processing, machine learning, communication theory, and information theory with applications to processing media, such as dynamic point clouds and meshes, video, images, audio, speech, and documents. He served as an Associate Editor of the IEEE TRANSACTIONS ON INFORMATION THEORY AND GUEST EDITOR for special issues in the IEEE TRANSACTIONS ON IMAGE PROCESSING, the IEEE TRANSACTIONS ON MULTIMEDIA, and the *IEEE Signal Processing Magazine*. He was a member of the IEEE Signal Processing Society (SPS) Image and Multidimensional Signal Processing Technical Committee, a member and Chair of the SPS Multimedia Signal Processing TC, a member of the ComSoc Multimedia TC, a member of the IEEE SPS and Computer Society Fellow Evaluation Committees, a member of the TMM and ICME Steering Committees, and a member of the SPS Board of Governors. He was the Founding Technical Chair of the inaugural NetCod 2005 workshop, the Special Session and Panel Chair for ICASSP 2007, the Publicity Chair of the Packet Video Workshop 2009, the Technical Co-Chair for MMSP 2009, the Awards Chair of ICIP 2015, and the Technical Program Co-Chair of ICME 2016 and ICIP 2017. He is a member of Phi Beta Kappa, Tau Beta Pi, and Sigma Xi, and the IEEE Computer, Information Theory, Signal Processing, and Communications societies, and was an Active Member of the MPEG Committee. He is a recipient, with T. Lookabaugh, of the 1993 Signal Processing Society Paper Award; with A. Seghal, of the 2002 ICME Best Paper Award; with Z. Miao, of the 2007 IEEE Transactions on Multimedia Best Paper Award; and with M. Ponec, S. Sengupta, M. Chen, and J. Li, of the 2009 ICME Best Paper Award, as well as co-author with H. Q. Nguyen of the 2014 ICASSP Best Student Paper Award (second place). He is a Co-Editor, with M. van der Schaar, of the book entitled *Multimedia over IP and Wireless Networks* (Elsevier, 2007).