A Volumetric Approach to Point Cloud Compression—Part II: Geometry Compression

Maja Krivokuća, Philip A. Chou, *Fellow, IEEE*, and Maxim Koroteev

***Abstract*— Compression of point clouds has so far been confined to coding the positions of a discrete set of points in space and the attributes of those discrete points. We introduce an alternative approach based on volumetric functions, which are functions defined not just on a finite set of points, but throughout space. As in regression analysis, volumetric functions are continuous functions that are able to interpolate values on a finite set of points as linear combinations of continuous basis functions. Using a B-spline wavelet basis, we are able to code volumetric functions representing both geometry and attributes. Attribute compression is addressed in Part I of this paper, while geometry compression is addressed in Part II. Geometry is represented implicitly as the level set of a volumetric function (the signed distance function or similar). Experimental results show that geometry compression using volumetric functions improves over the methods used in the emerging MPEG Point Cloud Compression (G-PCC) standard.**

***Index Terms*— Bézier volumes, B-splines, wavelets, point cloud compression, geometry coding, shape coding, multiresolution representations, signed distance function, graph signal processing.**

I. Introduction

THERE are two parts to the point cloud compression problem: geometry compression and attribute compression. In Part I of this paper [1], we addressed the problem of attribute compression: how to represent and compress attributes $f_1, \ldots, f_N$ on a given set of point locations $\boldsymbol{n}_1, \ldots, \boldsymbol{n}_N$. In that case, the point locations were assumed to be communicated out-of-band as side information and reconstructed at the decoder, either with or without loss. In the present part of the paper (Part II), we address the geometry compression problem: how to represent and compress the point locations themselves.

For geometry compression as well as attribute compression, we take a volumetric approach. In our volumetric approach, we fit a volumetric function to the data, the data being the attributes or geometry of a point cloud. By *volumetric function* we mean a real-valued function f defined throughout a volume of space. For geometry compression, as the volumetric function we typically choose a signed distance function (or similar, such as an occupancy probability or variant). The geometry is then represented *implicitly* as a level set (or solution set) of the volumetric function, i.e., $f^{-1}(c)$ for some real number c. This is in contrast to traditional *explicit* representations, such as lists of points or triangles. Although there exists prior work that represents geometry implicitly (e.g., [2], [3] and their derivatives), to our knowledge those works do not use the implicit representation for the purposes of compression. In contrast, in our volumetric approach, we compress the geometry by compressing the volumetric function that represents the geometry implicitly. To compress the volumetric function, we decompose the function into a linear combination of tri-linear B-spline wavelet basis functions, and then quantize and entropy-code the basis coefficients. Though other basis functions are possible, tri-linear B-spline wavelets are desirable because they are continuous and multiresolution.

This volumetric approach to geometry compression has several advantageous properties. First, it has excellent rate-distortion performance owing to the multiresolution nature of the wavelet basis. Over sufficiently small regions of space, the level set $f^{-1}(c)$ of a tri-linear B-spline f is nearly planar. Thus the granularity of the wavelet basis functions determines how well geometric surfaces are approximated. Large, nearly planar surfaces can be efficiently coded with very few coefficients, while detailed regions can be preserved by using more coefficients.

Second, as the tri-linear wavelet basis functions are continuous, any volumetric function that is a linear combination of such basis functions remains continuous, even after its basis coefficients are quantized. Since the level set $f^{-1}(c)$ of a continuous function f on $\mathbb{R}^3$ is a two-dimensional topological manifold having continuous charts, the reconstructed surface is guaranteed to have no "holes" even after quantization.

Third, and related to the second, is that one may zoom into the geometry ad infinitum without any gaps appearing.

Fourth, the signed distance function (or similar volumetric function) is a first step in many systems that reconstruct an explicit surface from many cameras. From a practical point of view, it may make more sense to compress the signed distance function directly, than to go through the extra steps of producing and compressing a triangulation, as the latter is less efficient and less amenable to multiresolution processing than the former.

Manuscript received October 16, 2018; revised August 19, 2019; accepted November 19, 2019. Date of publication December 11, 2019; date of current version January 10, 2020. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Damon M. Chandler. *(Corresponding author: Maja Krivokuća.)*

M. Krivokuća was with 8i, Wellington, New Zealand. She is now with the Inria Centre de Recherche Rennes Bretagne Atlantique, 35042 Rennes, France (e-mail: majakri01@gmail.com).

P. A. Chou was with 8i, Seattle, WA, USA. He is now with Google Inc Seattle, Seattle, WA 98103 USA.

M. Koroteev was with 8i, Wellington, New Zealand. He is now with Imatrex Inc., 141400 Moscow, Russia.

Digital Object Identifier 10.1109/TIP.2019.2957853

Experimental results show that our volumetric approach has better rate-distortion performance than the explicit point-based or triangle-based methods of geometry compression in the ongoing MPEG Point Cloud Compression (G-PCC) standardization activity [4]. Furthermore, its multiresolution nature allows scalable compression of the geometry and bitrates that are highly tunable.

The remainder of this paper is organized as follows. Section II applies the volumetric framework to geometry compression, Section III provides experimental results, and Section IV concludes the paper.

II. Geometry Coding Using Bézier Volumes

A. Implicit Surfaces

Let $f : \mathbb{R}^3 \to \mathbb{R}$ be a volumetric function and let c be a constant. The set of points $\boldsymbol{x}$ satisfying the equation $f(\boldsymbol{x}) = c$ is called the *level set* of f at level c, or the c-level set of f. The c-level set $\mathcal{S} = \{\boldsymbol{x} \,|\, f(\boldsymbol{x}) = c\}$ of f defines a set in $\mathbb{R}^3$ called an *implicit surface*.

Let Ω be a set in $\mathbb{R}^3$, and let $\mathcal{S} = \partial\Omega$ be its boundary.[1] Informally, we will refer to the boundary of a set as its *surface*. The *signed distance function* (SDF) of Ω is the function[2]

$$f(\boldsymbol{x}) = \begin{cases} -d(\boldsymbol{x}, \mathcal{S}) & \text{if } \boldsymbol{x} \in \Omega \\ d(\boldsymbol{x}, \mathcal{S}) & \text{otherwise,} \end{cases} \tag{1}$$

where $d(\boldsymbol{x}, \mathcal{S}) = \min_{\boldsymbol{x}_0 \in \mathcal{S}} ||\boldsymbol{x} - \boldsymbol{x}_0||$ is the distance from $\boldsymbol{x}$ to its closest point in the set $\mathcal{S}$.[3] In our work, $\mathbb{R}^3$ represents a Euclidean space, so the distance $||\boldsymbol{x} - \boldsymbol{x}_0||$ is a Euclidean distance. Clearly, $f(\boldsymbol{x}) = 0$ if and only if $\boldsymbol{x} \in \mathcal{S}$. Thus the boundary of Ω is given by the 0-level set of f,

$$\mathcal{S} = \{\boldsymbol{x} | f(\boldsymbol{x}) = 0\}. \tag{2}$$

If the function f can be approximated by a function $\hat{f}$, then the surface $\mathcal{S}$ can be approximated by the 0-level set of $\hat{f}$,

$$\widehat{\mathcal{S}} = \{\boldsymbol{x} | \hat{f}(\boldsymbol{x}) = 0\}. \tag{3}$$

Commonly, the surface $\mathcal{S}$ is locally planar. This means that near a point $\boldsymbol{x}_i$ on the surface, the signed distance function is approximated by the dot product $(\boldsymbol{x} - \boldsymbol{x}_i) \cdot \boldsymbol{n}_i$, where $\boldsymbol{n}_i$ is the unit normal to the surface at $\boldsymbol{x}_i$. Thus, near the surface, the signed distance function is nearly tri-linear and hence is well-approximated by a function in the subspace of tri-linear B-splines at a sufficiently high level of detail.

Our strategy is to approximate f by functions f_ℓ, $\ell = 0, 1, \ldots$ in a nested sequence of subspaces $\mathcal{F}_\ell$ [1, Eqn. 20] of tri-linear B-splines [1, Eqns. 91-93]. At a sufficiently high level of detail, say ℓ, the energy $||f - f_\ell||^2$ is low, meaning that the approximation f_ℓ is very close to f, as the surface is approximately planar over patches of diameter $2^{-\ell}$ or smaller. The parameters of f_ℓ can then be quantized, entropy-coded, and transmitted. The reconstructed function $\hat{f} = \hat{f}_\ell$, despite its quantized parameters, remains in $\mathcal{F}_\ell$, and hence is guaranteed to be continuous. Importantly, the reconstructed surface $\hat{\mathcal{S}}$ (3) is continuous in the sense that no holes develop as an artifact of quantization.

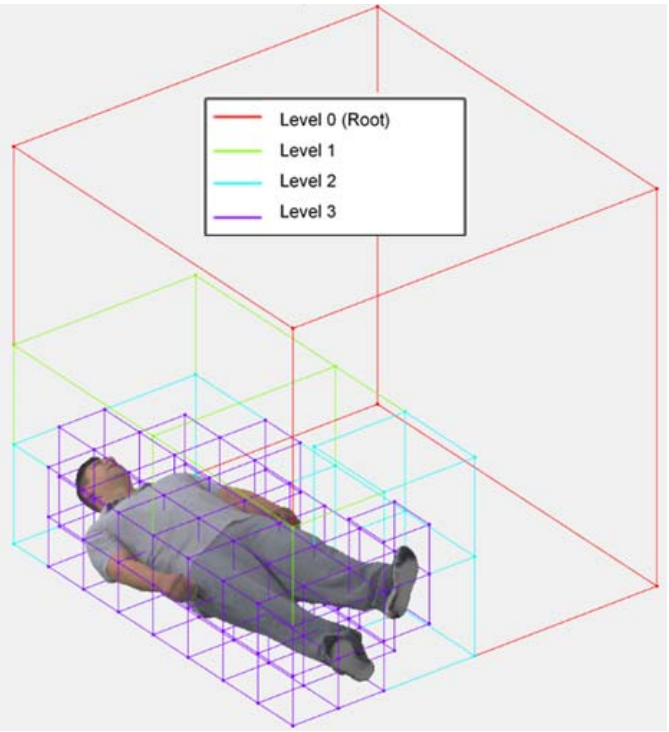


Fig. 1. Example of an octree decomposition of a 3D point cloud, up to level 3 (where the *root* (coarsest level) is considered level 0).

Computation of the function $f(\boldsymbol{x})$ is often an integral part of the processing pipeline used to produce the point cloud. For example, Curless and Levoy [2] compute a (truncated) signed distance function $f(\boldsymbol{x})$ of an object (from a set of depth camera measurements $\mathcal{M}$) in order to reconstruct the surface of the object as the 0-level set of f. Similarly, Loop *et al.* [6] compute the occupancy probability $p(\boldsymbol{x}) = P\{\boldsymbol{x} \in \Omega | \mathcal{M}\}$ in order to reconstruct the surface of an object as the (0.5)-level set of p, or equivalently, the 0-level set of the log odds $l(\boldsymbol{x}) = \log p(\boldsymbol{x})/(1 - p(\boldsymbol{x}))$.

In many cases, however, the function f may have to be derived from a finite collection of points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ sampled from a surface. In this case, the signed distance function at a point $\boldsymbol{x}$ can be approximated when $\boldsymbol{x}$ is far from the surface as

$$f(\boldsymbol{x}) = \text{sign}\left((\boldsymbol{x} - \boldsymbol{x}_{i^*}) \cdot \boldsymbol{n}_{i^*}\right) \left(\min_i ||\boldsymbol{x} - \boldsymbol{x}_i||\right), \tag{4}$$

where "$\cdot$" indicates the dot product and

$$i^* = \arg\min_i ||\boldsymbol{x} - \boldsymbol{x}_i||, \tag{5}$$

and can be approximated when $\boldsymbol{x}$ is close to a surface point $\boldsymbol{x}_i$ as

$$f(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{x}_i) \cdot \boldsymbol{n}_i, \tag{6}$$

where $\boldsymbol{n}_i$ is the unit normal at $\boldsymbol{x}_i$. Here we assume that a unit normal at each point can be computed. Our method for computing the SDF values (or *control points*)

[1] The *boundary* of a set in $\mathbb{R}^3$ is the set of points that are limit points both of the set and of its complement.

[2] Note that in (1) we use the SDF sign convention where a positive sign indicates that the point is outside the set and a negative sign indicates that the point is inside the set, following [5].

[3] Since $\mathcal{S}$ is closed, the minimum is well-defined and is achieved by a point $\boldsymbol{x}_0^* \in \mathcal{S}$.

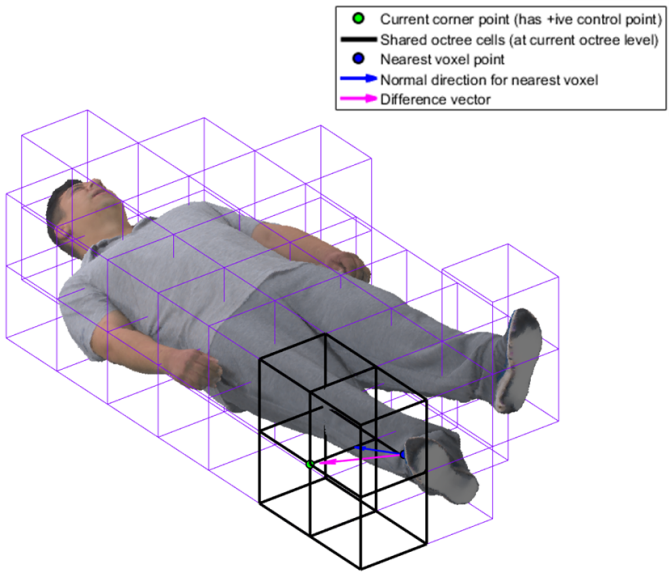


Fig. 2. Example of a positive SDF value (control point) on a shared octree block corner at level 3. The example corner, colored in green, is *outside* the point cloud surface, hence it has a positive SDF value.

follows (4)-(6). More specifically, we begin by first decomposing the input point cloud into an *octree* structure, which is a convenient spatial organization of the point cloud into 3D cubes, or *blocks*, at different resolution levels $\ell = 0, 1, \ldots$. (Part I of our paper [1] contains an introduction to octrees.) At each level ℓ, we consider only the *occupied* octree blocks (i.e., those containing a part of the point cloud "surface") for subdivision to the next, finer level. For example, see Fig. 1, which illustrates an octree decomposition of a 3D point cloud up to level 3. The SDF computation is then carried out at each octree level, by measuring the shortest distance to the point cloud "surface" (i.e., the distance to the nearest voxel) from each of the *unique*[4] corners of the occupied octree blocks at that level. We start from the octree *root* (level 0) and continue up to the *voxel* level (where the block size is $1 \times 1 \times 1$).[5] When checking for the nearest voxel to a corner, we consider only the voxels in the octree blocks (at the same octree level) that share that corner, e.g., see the shared octree blocks outlined in black for the corners shown in Figs. 2 and 3. We define a positive SDF value to indicate that the corresponding corner is *outside* the point cloud "surface", while a negative SDF value indicates that the corresponding corner is *inside* the surface, as in (1). As indicated in (4), a control point will be positive if the normal vector of the nearest voxel to that corner, and the difference vector between the corner and that nearest voxel, are pointing in a similar direction (i.e., the normal vector points *towards* the corner), while a control point will be negative if the normal vector and the difference vector point in different directions (i.e., the normal vector points *away* from the corner). Fig. 2 and Fig. 3 show examples of a positive and negative SDF value (control point), respectively, at octree level 3. Fig. 4 illustrates a positive control point computed at a coarser octree level (level 2). Note that the *difference vector* in Figs. 2 - 4 represents the vector $(\boldsymbol{x} - \boldsymbol{x}_{i^*})$ from (4), while the *normal vector* is the vector $\boldsymbol{n}_{i^*}$. Also note that the octree block outlined in black in Fig. 4 is the parent of the blocks outlined in black in Fig. 2. Since a finer octree level contains a greater number of blocks, there are more block corners

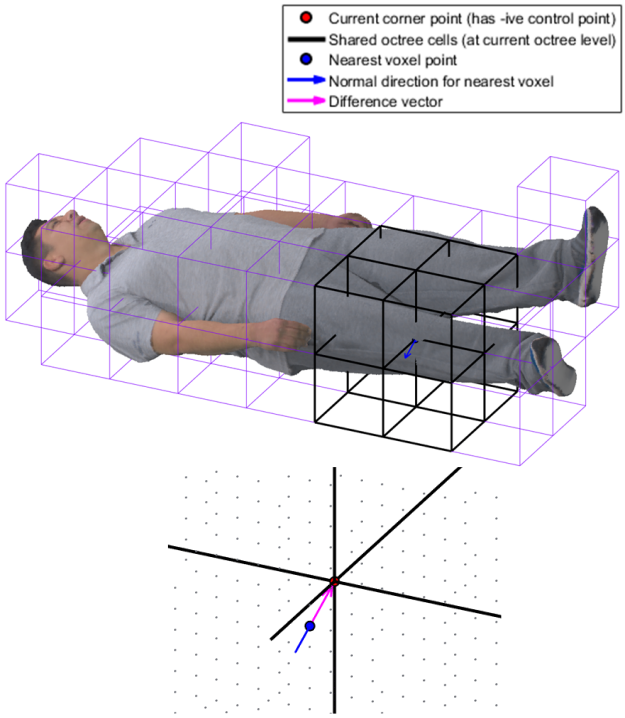


Fig. 3. Example of a negative SDF value (control point) on a shared octree block corner at level 3. The bottom image shows a zoomed-in version of the shared corner (which is found "inside" the man's trouser leg) and its nearest voxel indicated in the top picture, to illustrate that the difference vector between the corner and its nearest voxel is pointing in the opposite direction to the normal vector at that voxel, hence the control point for that corner is negative, as the corner is *inside* the point cloud surface.

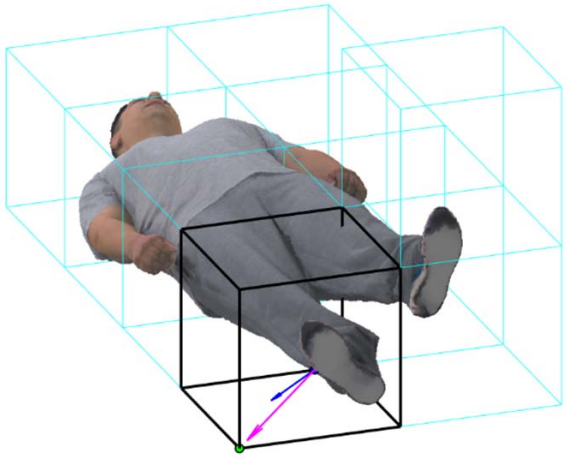


Fig. 4. Example of a positive SDF value (control point) on an octree block corner at level 2. The example corner, colored in green, is *outside* the point cloud surface, hence it has a positive SDF value.

[4]Since many of the corners are shared by neighboring octree blocks, we need to consider the SDF computation for each shared corner only once.

[5]A *voxel* (*vo*lumetric pi*xel*) is therefore an occupied octree block at the finest resolution level, d. At this finest resolution level, the point cloud is enclosed in a 3D cube (e.g., a regular 3D grid), which has a single spatial resolution of $2^d \times 2^d \times 2^d$ voxels.

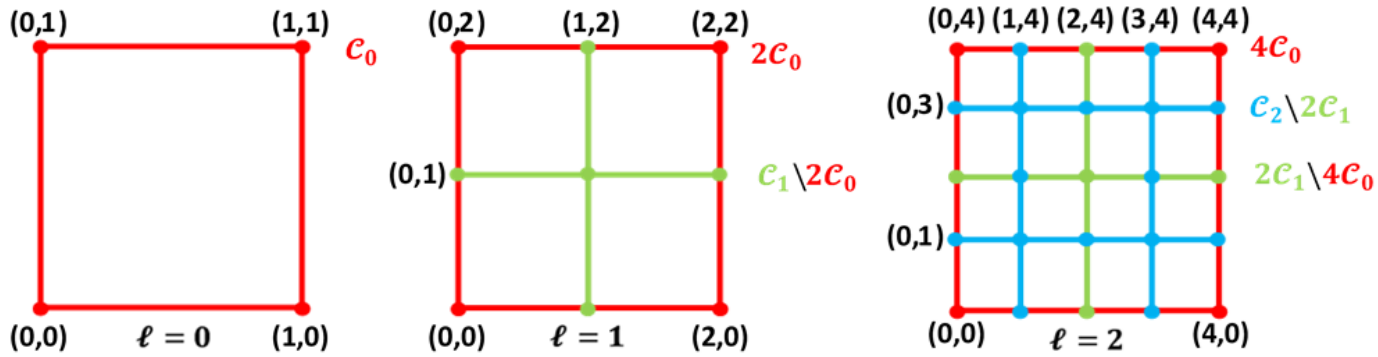


Fig. 5. An example of the nested blocks and corners in the proposed method, shown in 2D on a quadtree for different refinement levels ℓ. In our work, we apply this same notion to an octree in 3D. Note that the colors used for the different quadtree levels in this figure correspond to the colors used for the octree levels in Fig. 1, to facilitate understanding.

to compute control points from, and the distances between block corners and the point cloud "surface" are generally smaller (except for the corners that remain the same at all octree levels) since the blocks are smaller. Therefore, at finer octree levels, we obtain a finer sampling of the point cloud "surface", in terms of the number of control points that we compute, resulting in more precise reconstructions of the finer details in the point cloud's shape. An example of such a progressive shape reconstruction can be seen in Fig. 9. The mathematical theory behind this relates to the nested subspaces of the proposed tri-linear B-splines, which is explained in the following sub-section of this paper.

B. Nested B-Spline Spaces

As in [1, Eqn. 91], we begin with the central cardinal B-spline of order $p = 2$, or "hat" function,

$$\phi(x) = \begin{cases} 1 + x & x \in [-1, 0] \\ 1 - x & x \in [0, 1] \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

Let

$$\phi_{0,\mathbf{0}}(\boldsymbol{x}) = \phi(x)\phi(y)\phi(z) \tag{8}$$

be the volumetric version of the "hat" function, and let

$$\phi_{\ell,\boldsymbol{n}}(\boldsymbol{x}) = \phi_{0,\mathbf{0}}(2^{\ell}\boldsymbol{x} - \boldsymbol{n}) \tag{9}$$

be the "hat" function scaled to level ℓ and offset with vector integer shift $\boldsymbol{n}$. The space of piecewise tri-linear functions spanned by these functions at all shifts $\boldsymbol{n}$ at level ℓ is[6,7]

$$\mathcal{F}_\ell = \left\{ f_\ell \in \mathcal{F} \,\middle|\, \exists \{F_{\ell,\boldsymbol{n}}\} \text{ s.t. } f_\ell(\boldsymbol{x}) = \sum_{\boldsymbol{n} \in \mathcal{C}_\ell} F_{\ell,\boldsymbol{n}} \phi_{\ell,\boldsymbol{n}}(\boldsymbol{x}) \right\}. \tag{10}$$

Here $\mathcal{C}_\ell$ is the set of shifts indexing the corners of occupied blocks $\mathcal{B}_\ell$ at level ℓ. Note that $\mathcal{F}_\ell \subseteq \mathcal{F}_{\ell+1}$ because each function $\phi_{\ell,\boldsymbol{n}}(\boldsymbol{x})$ can be written as a linear combination of the functions $\{\phi_{\ell+1,\boldsymbol{k}},\ \boldsymbol{k} \in \mathbb{Z}^3\}$. Hence the subspaces are nested:

$$\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \ldots \subseteq \mathcal{F}_\ell \subseteq \mathcal{F}_{\ell+1} \subseteq \ldots \subseteq \mathcal{F}. \tag{11}$$

Since $\phi_{\ell,\boldsymbol{n}}(2^{-\ell}\boldsymbol{n}) = 1$ and $\phi_{\ell,\boldsymbol{n}}(2^{-\ell}\boldsymbol{k}) = 0$ for all $\boldsymbol{k} \neq \boldsymbol{n}$, $\boldsymbol{k} \in \mathbb{Z}^3$, it follows that $F_{\ell,\boldsymbol{n}} = f_\ell(2^{-\ell}\boldsymbol{n})$ for all $\boldsymbol{n} \in \mathbb{Z}^3$. Hence the coefficients representing f_ℓ in the basis $\{\phi_{\ell,\boldsymbol{n}}\}$ are simply the values of $f_\ell(\boldsymbol{x})$ at the corners $\mathcal{C}_\ell$.

At this point, we could develop the usual decomposition of $\mathcal{F}_{\ell+1}$ into $\mathcal{F}_\ell$ and its orthogonal complement $\mathcal{G}_\ell$, where $\mathcal{G}_\ell$ is spanned by wavelet basis functions that are orthogonal not only to $\mathcal{F}_\ell$ but to each other. However, these wavelet basis functions would have infinite support, and would be impractical to implement in $\mathbb{R}^3$.

We propose, instead, a non-orthogonal decomposition $\mathcal{F}_{\ell+1} = \mathcal{F}_\ell \oplus \mathcal{G}_\ell$, where

$$\mathcal{G}_\ell = \left\{ g_\ell \in \mathcal{F} \,\middle|\, \exists \{G_{\ell,\boldsymbol{n}}\} \text{ s.t. } g_\ell = \sum_{\boldsymbol{n} \in \mathcal{C}_{\ell+1} \backslash 2\mathcal{C}_\ell} G_{\ell,\boldsymbol{n}} \phi_{\ell+1,\boldsymbol{n}} \right\}. \tag{12}$$

Clearly, $\mathcal{G}_\ell$ is a subspace of $\mathcal{F}_{\ell+1}$ spanned by the functions $\phi_{\ell+1,\boldsymbol{n}}$ centered at the corners of the blocks at level $\ell + 1$ that are not also corners of any blocks at level ℓ, namely $\{\phi_{\ell+1,\boldsymbol{n}} : \boldsymbol{n} \in \mathcal{C}_{\ell+1} \setminus 2\mathcal{C}_\ell\}$.[8] Just as $F_{\ell,\boldsymbol{n}} = f_\ell(2^{-\ell}\boldsymbol{n})$ in (10), $G_{\ell,\boldsymbol{n}} = g_\ell(2^{-(\ell+1)}\boldsymbol{n})$ in (12), for all $\boldsymbol{n} \in \mathbb{Z}^3$. Thus $\mathcal{G}_\ell$ is the subspace of functions in $\mathcal{F}_{\ell+1}$ that are zero on the corners of the blocks in $\mathcal{B}_\ell$. A simple graphical example of the nested blocks and corners is illustrated on a quadtree in 2D, in Fig. 5, for quadtree levels $\ell = 0$ to $\ell = 2$. We see in Fig. 5 that with each quadtree refinement, the newly introduced corners $\mathcal{C}_{\ell+1} \setminus 2\mathcal{C}_\ell$ at level $\ell + 1$ exclude the corners $\mathcal{C}_\ell$ (after scaling by two) already introduced at previous levels, but together all of these corners constitute the corners at level $\ell + 1$. The newly introduced corners $\mathcal{C}_{\ell+1} \setminus 2\mathcal{C}_\ell$ are the corners on which the functions $\phi_{\ell+1,\boldsymbol{n}}$ in $\mathcal{G}_\ell$ are centered, and the other corners $2\mathcal{C}_\ell$ at level $\ell + 1$ are the corners on which these functions (and hence functions $g_\ell \in \mathcal{G}_\ell$) are zero. In our work, this same notion is extended to an octree in 3D, where each parent block is subdivided into 8 child blocks instead

[6]Here $\mathcal{F}_\ell$ is defined as in [1, Eqn. 19], where $\mathcal{F}$ is the Hilbert space under the inner product with respect to the counting measure [1, Eqn. 1] on the points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$.

[7]Note that the operator $\exists$ here denotes an existential quantifier, which can be interpreted as "there exists".

[8]Note that the backslash operator here denotes a set difference; that is, $A \setminus B = \{x : x \in A \text{ and } x \notin B\}$.

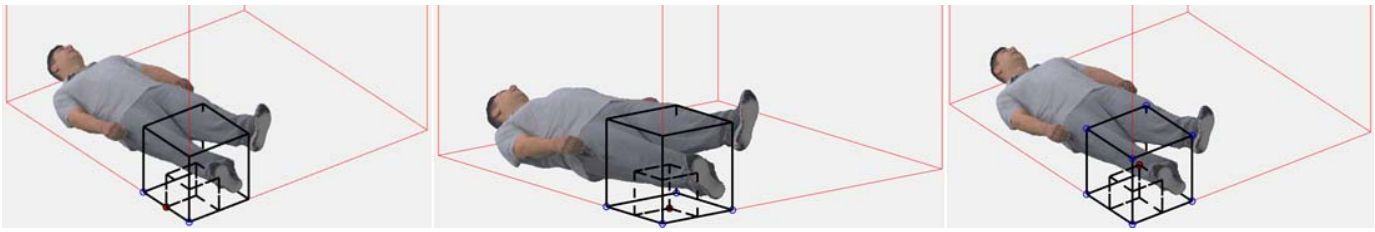


Fig. 6. Examples of parent corners at level ℓ used to obtain predictions for child corners at level $\ell+1$. The example child corners are colored in red, the corresponding parent corners are circled in blue, and part of the root octree block (red) is shown for reference. (Left) The child corner is on a parent block *edge*; (Middle) The child corner is on a parent block *face*; (Right) The child corner is in the *center* of the parent block.

of 4. Note that, as mentioned in Section II-A, we only further subdivide the *occupied* octree blocks at each level, but for the sake of simplicity this is not shown on the quadtree in Fig. 5.

It can be seen that if $f_\ell \in \mathcal{F}_\ell$ and $g_\ell \in \mathcal{G}_\ell$ then $f_\ell + g_\ell$ is a function $f_{\ell+1} \in \mathcal{F}_{\ell+1}$ such that for all $\boldsymbol{n} \in 2\mathcal{C}_\ell$, $f_{\ell+1}(2^{-(\ell+1)}\boldsymbol{n}) = f_\ell(2^{-\ell}\boldsymbol{n}/2) = F_{\ell,\boldsymbol{n}/2} = F_{\ell+1,\boldsymbol{n}}$, and for all $\boldsymbol{n} \in \mathcal{C}_{\ell+1} \setminus 2\mathcal{C}_\ell$, $f_{\ell+1}(2^{-(\ell+1)}\boldsymbol{n}) = f_\ell(2^{-(\ell+1)}\boldsymbol{n}) + g_\ell(2^{-(\ell+1)}\boldsymbol{n}) = f_\ell(2^{-(\ell+1)}\boldsymbol{n}) + G_{\ell,\boldsymbol{n}} = F_{\ell+1,\boldsymbol{n}}$. Thus for all $\boldsymbol{n} \in \mathcal{C}_{\ell+1} \setminus 2\mathcal{C}_\ell$, $f_\ell(2^{-(\ell+1)}\boldsymbol{n})$ can be considered the *prediction* of $f_{\ell+1}(2^{-(\ell+1)}\boldsymbol{n})$, and $G_{\ell,\boldsymbol{n}}$ can be considered the *prediction error* or *residual*. Thus $\mathcal{F}_{\ell+1}$ is the direct sum of $\mathcal{F}_\ell$ and $\mathcal{G}_\ell$.

This implies

$$\mathcal{F}_{\ell+1} = \mathcal{F}_0 \oplus \mathcal{G}_0 \oplus \mathcal{G}_1 \oplus \cdots \oplus \mathcal{G}_\ell, \tag{13}$$

and hence any function $f_{\ell+1} \in \mathcal{F}_{\ell+1}$ can be written

$$f_{\ell+1} = f_0 + g_0 + g_1 + \cdots + g_\ell, \tag{14}$$

where the coefficients $\{F_{0,\boldsymbol{n}}\}$ of f_0 in the basis for $\mathcal{F}_0$ are *low-pass* coefficients, while the coefficients $\{G_{\ell,\boldsymbol{n}}\}$ of g_ℓ in the basis for $\mathcal{G}_\ell$ are *high-pass* or *wavelet* coefficients.

Any function $f : \mathbb{R}^3 \to \mathbb{R}$ can be projected onto $\mathcal{F}_\ell$ by sampling:

$$F_{\ell,\boldsymbol{n}} = f_\ell(2^{-\ell}\boldsymbol{n}) = f(2^{-\ell}\boldsymbol{n}) \tag{15}$$

for all $\boldsymbol{n} \in \mathbb{Z}^3$. That is, the projection of f onto $\mathcal{F}_\ell$, denoted $f_\ell = f \circ \mathcal{F}_\ell$, interpolates f at the knots $2^{-\ell}\mathbb{Z}^3$. Note that this definition of projection is not the least-squares projection of f onto $\mathcal{F}_\ell$ under the inner product. However, under this definition of projection, the functions $g_\ell \in \mathcal{G}_\ell$ are orthogonal to $\mathcal{F}_\ell$, in the sense that they are zero at the knots $2^\ell\mathbb{Z}^3$ and thus project to the zero function in $\mathcal{F}_\ell$.

Fig. 6 shows examples of how we compute the SDF (control point) predictions for child corners at level $\ell+1$, depending on their location on the corresponding parent octree block at level ℓ, i.e., whether the child corner is on a parent block *edge*, *face*, or in the parent block *center*. The *prediction* is equal to the average of the SDF values of the parent corners circled in blue in Fig. 6, for the corresponding child corner colored in red in the same diagram. The *prediction error*, or *residual*, or *wavelet coefficient*, for that child corner is then the difference between the child's SDF value and the prediction (average of the parent control points).[9]

C. Octree Coding of Bézier Volumes

To compress an implicit surface $\{\boldsymbol{x} : f(\boldsymbol{x}) = c\}$, we compress f to a function $\hat{f}$ and represent the surface as $\{\boldsymbol{x} : \hat{f}(\boldsymbol{x}) = c\}$, where $c = 0$.

The straightforward way to compress f would be to project it to a function f_ℓ with sufficiently high level of detail $\ell = d$, represent f_d as the sum of functions $f_0 + g_0 + g_1 + \cdots + g_{d-1}$, and then quantize and entropy-code the coefficients $F_{0,\boldsymbol{n}}$ (for all $\boldsymbol{n} \in \mathcal{C}_0$), $G_{0,\boldsymbol{n}}$ (for all $\boldsymbol{n} \in \mathcal{C}_1 \setminus 2\mathcal{C}_0$), $G_{1,\boldsymbol{n}}$ (for all $\boldsymbol{n} \in \mathcal{C}_2 \setminus 2\mathcal{C}_1$), and so forth up through $G_{d-1,\boldsymbol{n}}$ (for all $\boldsymbol{n} \in \mathcal{C}_d \setminus 2\mathcal{C}_{d-1}$).

However, because the subspaces $\mathcal{F}_0, \mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_{\ell-1}$ are not orthogonal to each other (in the least-squares sense), in this straightforward approach the norm of the quantization error would be larger in the reconstructed domain than in the coefficient domain, due to error propagation.

To mitigate the problem of error propagation, f is compressed sequentially with quantization in the loop as follows. Initially, f is projected onto the subspace $\mathcal{F}_0$ by sampling f on the corners of the unit cube, $F_{0,\boldsymbol{n}} = f_0(\boldsymbol{n}) = f(\boldsymbol{n})$ (for all $\boldsymbol{n} \in \mathcal{C}_0$). These low-pass coefficients are uniformly scalar-quantized with stepsize Δ, entropy-coded, and transmitted. Let $\hat{F}_{0,\boldsymbol{n}} = \hat{f}_0(\boldsymbol{n}) = \hat{f}(\boldsymbol{n})$ (for all $\boldsymbol{n} \in \mathcal{C}_0$) denote their reconstruction. Then, for $\ell = 0, 1, \ldots, d-1$ in sequence, the prediction $\hat{f}_\ell(2^{-(\ell+1)}\boldsymbol{n})$ is formed by tri-linear interpolation of $\hat{f}_\ell$ (for all $\boldsymbol{n} \in \mathcal{C}_{\ell+1}$); f is projected onto the subspace $\mathcal{F}_{\ell+1}$ as $F_{\ell+1,\boldsymbol{n}} = f_{\ell+1}(2^{-(\ell+1)}\boldsymbol{n}) = f(2^{-(\ell+1)}\boldsymbol{n})$ (for all $\boldsymbol{n} \in \mathcal{C}_{\ell+1}$); the wavelet coefficients $G_{\ell,\boldsymbol{n}} = F_{\ell+1,\boldsymbol{n}} - \hat{f}_\ell(2^{-(\ell+1)}\boldsymbol{n})$ are uniformly scalar-quantized with stepsize Δ, entropy-coded, transmitted, and recovered as $\hat{G}_{\ell,\boldsymbol{n}}$ (for all $\boldsymbol{n} \in \mathcal{C}_{\ell+1} \setminus 2\mathcal{C}_\ell$); and the quantized low-pass coefficients at level $\ell+1$ are recovered as $\hat{F}_{\ell+1,\boldsymbol{n}} = \hat{G}_{\ell,\boldsymbol{n}} + \hat{f}_\ell(2^{-(\ell+1)}\boldsymbol{n})$ (for all $\boldsymbol{n} \in \mathcal{C}_{\ell+1}$). This guarantees that

$$\begin{aligned} &|f_{\ell+1}(2^{-(\ell+1)}\boldsymbol{n}) - \hat{f}_{\ell+1}(2^{-(\ell+1)}\boldsymbol{n})| \\ &\quad = |F_{\ell+1,\boldsymbol{n}} - \hat{F}_{\ell+1,\boldsymbol{n}}| = |G_{\ell+1,\boldsymbol{n}} - \hat{G}_{\ell+1,\boldsymbol{n}}| \le \Delta/2 \end{aligned} \tag{16}$$

[9]Note that our proposed approach should not be confused with the work in [7], which organizes a 3D point cloud into a 2D grid, then hierarchically splits the 2D grid such that Bézier patches can be fit to approximate the shape of the surface corresponding to the 3D points in each 2D quadrant. The 3D point locations are then compressed by encoding the set of Bézier control points in each quadrant and the residuals that represent the differences between the original 3D point locations and their corresponding points on the patches.

for all ℓ and $\boldsymbol{n} \in \mathbb{Z}^3$, thus guaranteeing that errors do not propagate.

If f is the signed distance function, then (16) guarantees that the location of the reconstructed surface $\{\boldsymbol{x} : \hat{f}(\boldsymbol{x}) = 0\}$ is within Hausdorff distance $\Delta/2$ of the original surface $\{\boldsymbol{x} : f(\boldsymbol{x}) = 0\}$.

D. Pruning the Octree Before Encoding

It is usually not necessary to transmit the entire octree and all of its associated wavelet coefficients to the decoder, particularly in the case of *lossy* shape compression. One reason for this is that many of the quantized wavelet coefficients will be zero, or near zero, close to the leaves (voxel level) of the octree, or wherever the surface is locally flat (or nearly flat). This means that the octree blocks with such small wavelet coefficients will not contribute much to the overall quality of shape reconstruction of the 3D object. We therefore experimented with several different methods for octree pruning, all of which fit into the following two categories:

- **Fixed-level pruning:** All the octree blocks and their associated wavelet coefficients beyond one chosen octree level are pruned off. In this case, all the octree *leaves* (blocks with no descendants) end up being at the same octree level after pruning.
- **Variable-level pruning:** Octree blocks and their associated wavelet coefficients are pruned according to some rate-distortion algorithm or other criteria applied on different branches of the octree, so that after pruning the octree leaves can be at variable octree levels.

The octree *leaves* that remain after pruning in any of the methods described above are termed the *Bézier Volumes* (BVs) in our work.

For the case of variable-level pruning, we also tried a number of different solutions, mainly:

1) Pruning Based on Zero Wavelet Coefficients: When the wavelet coefficients are zero on the corners of all of a block's descendant blocks, then the octree may be pruned at that ancestral block, leaving the ancestral block as a leaf, or *Bézier Volume* (BV) of order $p = 2$, so-called because the value of $\hat{f}$ within the block can be completely determined by tri-linear interpolation of the values of the low-pass coefficients, or *control points*, at the corners of the BV block.[10] Thus the surface $\{\boldsymbol{x} : \hat{f}(\boldsymbol{x}) = 0\}$ within the block is likewise determined by tri-linear interpolation of the control points on the corners of the block. Furthermore, because neighboring blocks (even if they are at different levels) share the same values on any shared corners, they share the same values on any shared face; hence they share the same level set on any shared face; hence the reconstructed surface between neighboring blocks is continuous regardless of the quantization stepsize Δ.

2) Pruning Based on Zero and Small Non-Zero Wavelet Coefficients: This is similar to the method described above, except that instead of only considering quantized wavelet coefficients with values of zero, we also consider quantized wavelet coefficients with other "small" (within some chosen threshold) non-zero values. This effectively corresponds to applying a deadzone to the uniform scalar quantizer that is used to quantize the wavelet coefficients.

3) Pruning Based on Estimated Geometry Reconstruction Error: The descendants of octree blocks in which the estimated geometry reconstruction error is "small enough" (i.e., within some chosen error threshold) are pruned off, along with their associated wavelet coefficients.

4) Pruning Based on Rate-Distortion Optimization: Instead of considering only the estimated geometry reconstruction error as described in the method above, in this case we also consider the estimated changes in bitrate if an octree block and its descendants are pruned versus not pruned. This method is based on the optimal rate-distortion pruning theory described in [8].

The experimental results and discussions on the successes and failures of the pruning methods described above are provided in Section III.

E. Reconstructing a Point Cloud From the Implicit Surface

The ability to reconstruct a volumetric function $\hat{f}$ whose level set represents the surface implicitly is not usually the end result. In systems of practical interest, the surface must be made explicit. Typically, this means rasterizing or voxelizing the implicit surface into a finite set of points. This can be considered *rendering* or *reconstructing* an explicit point cloud from the implicit surface.

In this subsection, we outline two methods in which the implicit surface within a Bézier Volume can be made explicit.

In the first method, known as *recursive subdivision*, the Bézier Volume is recursively subdivided to a particular level. We say that a block at level ℓ has a *c-crossing* if $\min\{\hat{F}_{\ell,\boldsymbol{n}}\} \leq c \leq \max\{\hat{F}_{\ell,\boldsymbol{n}}\}$, where $\{\hat{F}_{\ell,\boldsymbol{n}}\}$ is the set of reconstructed control points at all eight corners $\boldsymbol{n}$ of this block, and the min and max are computed over all of these control points. That is, some reconstructed control points lie on one side of c, and some lie on the other. If a block has a c-crossing, then it contains a part of the implicit surface $\{\boldsymbol{x} : \hat{f}(\boldsymbol{x}) = c\}$, and hence is *occupied*. An occupied block at level $\ell = d$ (where d is the octree depth) is declared to be an occupied *voxel*, while an occupied block at level $\ell < d$ is subdivided into eight sub-blocks at level $\ell+1$, and the process is repeated on each occupied sub-block after computing the control points of the sub-block using tri-linear interpolation of the control points of the parent block. This procedure is guaranteed to find all occupied voxels.

In the second method, known as *raycasting*, the Bézier Volume is raster-scanned down the dominant axis of the surface within the BV. The dominant axis is the axis closest to the surface normal within the BV. We estimate the surface normal by taking the gradient of the volumetric function $\hat{f}$ within the BV. Assume the BV is normalized to the unit cube $[0, 1]^3$, and that the control points at its eight corners are $\hat{F}(i, j, k)$, $i, j, k \in \{0, 1\}$. Then at any point (x, y, z) within

[10] Refer to our definition of a Bézier Volume in [1, Section III-C].

TABLE I
DATASETS USED FOR GEOMETRY REPRESENTATION AND COMPRESSION

Dataset name	Original resolution	Original point count
boxer_8i_vox10	10 bit	993745
longdress_1300_8i_vox10	10 bit	911432
loot_1200_8i_vox10	10 bit	868658
soldier_0690_8i_vox10	10 bit	1191745

the BV, the value of $\hat{f}(x, y, z)$ is tri-linearly interpolated as

$$\hat{f}(x, y, z) = \sum_{i=0}^{1}\sum_{j=0}^{1}\sum_{k=0}^{1} F(i, j, k) \times x^i(1-x)^{1-i} y^j(1-y)^{1-j} z^k(1-z)^{1-k}. \quad (17)$$

Thus the gradient within the BV has constant components

$$\frac{\partial \hat{f}}{\partial x} = \sum_{j=0}^{1}\sum_{k=0}^{1}[F(1, j, k) - F(0, j, k)] \quad (18)$$

$$\frac{\partial \hat{f}}{\partial y} = \sum_{i=0}^{1}\sum_{k=0}^{1}[F(i, 1, k) - F(i, 0, k)] \quad (19)$$

$$\frac{\partial \hat{f}}{\partial z} = \sum_{i=0}^{1}\sum_{j=0}^{1}[F(i, j, 1) - F(i, j, 0)]. \quad (20)$$

The component whose absolute value is the largest determines the dominant axis. Say the dominant axis is z. Then the (x, y) plane is rasterized into a finite set of points $(x_m, y_m) \in [0, 1]^2$, and for each (x_i, y_i) the intersection with the implicit surface by the ray through (x_i, y_i) along z is determined as

$$z_m = \left[\sum_{i=0}^{1}\sum_{j=0}^{1} F(i, j, 0) x_m^i (1-x_m)^{1-i} y_m^j (1-y_m)^{1-j}\right] / \left[\sum_{i=0}^{1}\sum_{j=0}^{1} F(i, j, 0) x_m^i (1-x_m)^{1-i} y_m^j (1-y_m)^{1-j} - \sum_{i=0}^{1}\sum_{j=0}^{1} F(i, j, 1) x_m^i (1-x_m)^{1-i} y_m^j (1-y_m)^{1-j}\right]. \quad (21)$$

The collection of points $\{(x_m, y_m, z_m)\}$ is thus a rasterization of the implicit surface.

III. EXPERIMENTAL RESULTS

The results presented in this section relate to the datasets in Table I. All of the point clouds in Table I have 10-bit geometry resolution, meaning that the point (x, y, z) locations are integers in the range [0, 1023]. The *longdress*, *loot*, and *soldier* datasets are similar to the corresponding datasets used by the MPEG Point Cloud Coding (PCC) group, except that the normals have been recomputed to be more accurate and the point count is slightly different. The 10-bit version of the *boxer* dataset is not currently used by the MPEG PCC group.

Fig. 7 shows the rate-distortion performance of the BV geometry compression method versus the Trisoup (or S-PCC) method that is the Test Model for geometry compression of static (single-frame) point clouds in the emerging MPEG G-PCC standard [4]. The proposed BV method can be seen as an alternative method to Trisoup, of fitting a surface to

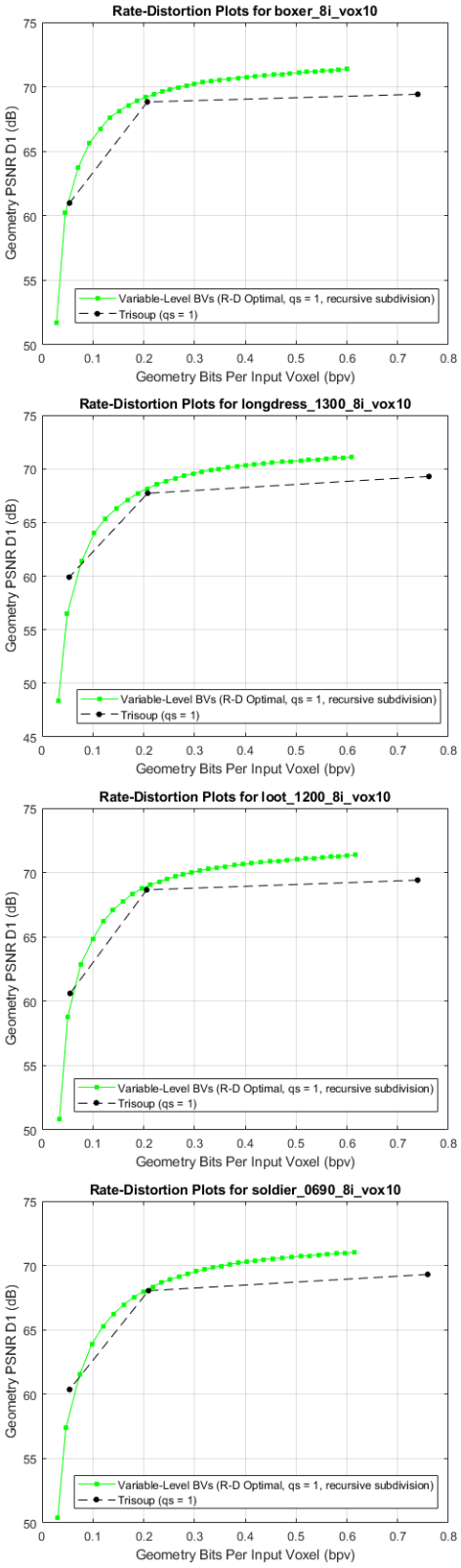


Fig. 7. Rate-distortion performance for BV vs Trisoup, when BV uses rate-distortion optimal pruning and recursive subdivision for voxel reconstruction.

the occupied leaf blocks of an octree, to more efficiently represent the point locations within these blocks. (In fact, the possibility of such an alternative method is mentioned in [4, Section VI-B].) In Trisoup, the surface is represented and coded *explicitly* as a collection of triangles in each leaf block of an octree (currently in Trisoup, all the leaves are at the same octree level, which is chosen by the user). The intersections of the surface with the edges of octree blocks are called *vertices* and are encoded using two pieces of data: (i) a bit vector that indicates which edges contain a surface intersection and which do not, and (ii) for each edge that contains a vertex, the position of that vertex along the edge. The vertex positions are quantized, and then the bit vector, the quantized vertex positions, and the octree occupancy bytes are further compressed by an entropy coder.

For the BV method in Fig. 7, the *rate-distortion optimal* pruning method is used (see Section II-D), and the *recursive subdivision* method (see Section II-E) is used to reconstruct the voxels from the implicit surface.

The *distortion* in Fig. 7 is the point-to-point geometry PSNR obtained from the *pc_error* measurement tool developed by the MPEG PCC group [9] (version 0.10 of pc_error was used to obtain the geometry PSNR results in this paper). The "qs = 1" for both the BV and Trisoup methods in Fig. 7 indicates that a uniform scalar quantization stepsize of 1 was used for all the transmitted geometry-related data that was quantized.

The *rate* in the plots in Fig. 7 represents the geometry bitrate measured as bits per input voxel of the corresponding input point cloud, where the number of input voxels is as shown in Table I. Included in the geometry bitrate for BV are the following:

- Occupancy codes of the pruned octree, written to a file using 8 bits per internal (non-leaf) octree node,
- A "post-pruning array" that indicates which of the octree blocks that remain after pruning are leaves (represented by a "1" bit) and which are internal (represented by a "0" bit), packed into bytes and written to a file,
- Control points (SDF values) at a chosen coarse level ("start level") of the octree (level 2 was chosen for the results presented in this paper), uniformly scalar-quantized and written to a file,
- Wavelet coefficients at each octree level, from the next level after "start level" up to and including the variable leaf levels (excluding the case where the leaves are at the voxel level, as these voxel positions are simply reconstructed from the occupancy codes and so do not need their SDFs to be reconstructed at the decoder), uniformly scalar-quantized and rANS entropy-encoded [10] by octree level, then further compressed (i.e., the individual binary files produced by rANS for each octree level are consolidated into a single binary file) using 7-zip (7z format) [11].

The individual files obtained above are then all added to the same 7z [12] archive and consolidated into one binary 7z file. The size of this consolidated binary file has been used to obtain the BV geometry bitrates presented in this paper. 7z has many compression algorithms to choose from [12], but we use the default, LZMA. While it might be worthwhile to investigate other lossless compression algorithms in the future, a different algorithm is unlikely to significantly alter the results, for two main reasons. Firstly, the LZ* algorithms are asymptotically optimal, converging to the entropy rate of any underlying stationary ergodic process [13], [14]. Secondly, in the proposed BV method, by far the largest portion of the total bitrate is used for the wavelet coefficients and the occupancy codes, as shown in the example in Table II. The wavelet coefficients are already entropy-coded with rANS prior to using 7z, so any further compression is minor. The octree occupancy codes appear to be nearly independently distributed, so any further compression beyond first-order entropy coding would also be minor, were a simple first-order entropy coder to be used. The latter observation also applies to the post-pruning array and the start-level control points, but these contribute very little to the final bitrate anyway, as shown in Table II.

TABLE II
DISTRIBUTION OF GEOMETRY BITS FOR *boxer_8i_vox10*, FOR THE BV RESULTS SHOWN IN FIG. 7, AT THE POINT WHERE A GEOMETRY PSNR (D1) OF AROUND 71 dB IS REACHED AND THE TOTAL GEOMETRY BITRATE IS APPROXIMATELY 0.47 BPV. NOTE THAT THE BPV VALUES PRESENTED IN THIS TABLE HAVE BEEN TAKEN FROM THE CORRESPONDING INVIDIDUAL FILE SIZES IN THE 7Z ARCHIVE *prior to packing*, SO ADDING THEM UP GIVES A TOTAL BPV OF 0.52. THIS DEMONSTRATES THAT THE 7Z COMPRESSION OF THE ARCHIVE DOES NOT SIGNIFICANTLY CHANGE THE FINAL OVERALL BITRATE, AS THE PACKED SIZE OF THE ARCHIVE (0.47 BPV) IS ONLY 0.05 BPV SMALLER

Encoded data	No. of bits used (bpv)
Occupancy codes	0.21
Wavelet coefficients	0.23
Post-pruning array	0.080
Start-level control points	0.00055
Sum of individual file sizes in 7z archive	64,453 Bytes (= 0.52 bpv)
Packed size of 7z archive	58,390 Bytes (= 0.47 bpv)

Fig. 7 demonstrates that the BV with R-D-optimal pruning and recursive subdivision voxel reconstruction outperforms Trisoup at all, or almost all, bitrates, and achieves near-maximum geometry PSNR (around 70 dB) at less than half the bitrate that Trisoup requires to get close to that quality level (around 69 dB). We have found that using this pruning method, combined with the recursive subdivision voxel reconstruction method, generally produces the best R-D results.

Fig. 8 additionally compares the rate-distortion performance of the BV method when using the different pruning methods described in Section II-D, combined with either the *recursive subdivision* or the *raycasting* method for voxel reconstruction (described in Section II-E). The results in Fig. 8 are generally representative of the R-D results for all 4 datasets in Table I.

Several observations may be made from the results in Fig. 8. Firstly, it is clear that the *recursive subdivision* voxel reconstruction method usually leads to significantly better PSNR results than when *raycasting* is used with the same pruning method. The wider raycasting range of [−2/BV_side_length, 1 + 2/BV_side_length] in Fig. 8, where *BV_side_length* represents the width of an octree leaf block (or BV), indicates that here the voxel reconstruction was allowed to go slightly

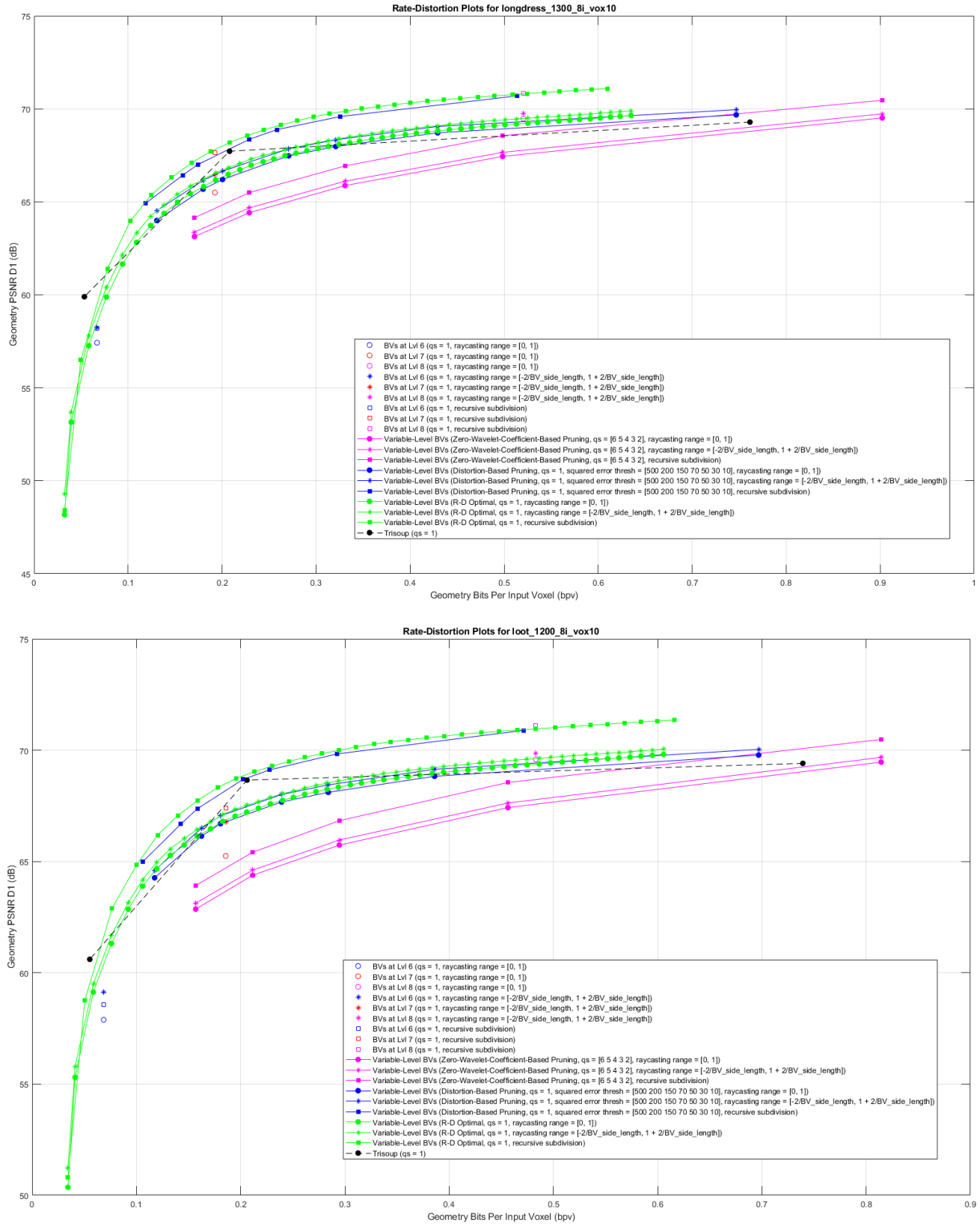


Fig. 8. Rate-distortion performance for BV vs Trisoup, when different pruning methods and voxel reconstruction methods are used for BV.

outside the BV blocks (i.e., outside the usual raycasting range of [0, 1] (see Section II-E)). We believe that the reason the subdivision method outperforms raycasting, and the raycasting with a wider range outperforms the raycasting with the [0, 1] range, is that the former two methods produce a thicker voxelized surface. This potentially allows for more accurate

nearest neighbors to be found during the point-to-point PSNR computation in pc_error, which matches input and output voxels based on a nearest-neighbor computation. However, we have also found that increasing the raycasting range beyond about [−2.5/BV_side_length, 1 + 2.5/BV_side_length], while still improving the PSNR slightly over the lower raycasting ranges (but still not better than when using recursive subdivision), tends to produce a voxelized surface that is noticeably too thick in certain places, at least in the kinds of point clouds that we tested on. That is, the extra voxels noticeably protrude out from the rest of the surface, which is visually unappealing. This does not happen in the recursive subdivision case, which still achieves superior PSNR results.

Fig. 8 also shows that, as expected, the rate-distortion optimal pruning usually produces better results than the corresponding fixed-level pruning when using the same voxel reconstruction method. However, it can sometimes happen that the fixed-level pruning has a slightly better PSNR, for example the fixed-level pruning at level 8 using raycasting is slightly better than the corresponding rate-distortion optimal pruning for *longdress* and *loot* in Fig. 8. We believe that this can be attributed to slight inaccuracies in the distortion and/or rate estimations during the rate-distortion optimal pruning.

Fig. 9 shows examples of the geometry (shape) reconstructions obtained for *boxer_8i_vox10*, when *fixed-level* BV pruning is used with the *recursive subdivision* voxel reconstruction method. We see that increasing the pruning level results in a progressively more refined shape reconstruction, and that pruning beyond octree level 8 (i.e., the BV blocks are at level 8) tends to produce an almost perfect geometry reconstruction, as confirmed by the PSNR results in Fig. 8. Choosing a pruning level that is too small can result in BV blocks that are too big for the surface that fits inside them, so all of the control points on the corners of these BVs would have the same sign (+ in our implementation) since these corners are all outside the surface. This means that, even though such a BV would be occupied, it would not contain a zero crossing, so it would not be subdivided during voxel reconstruction at the decoder; therefore, there would be no reconstructed surface (voxels) inside that block. For example, this happens in the octree blocks at level 3 in Fig. 9, which contain the man's feet and his head. This is why for the pruning at level 3, the man has no head or feet (amongst other missing features).

Probably the most surprising observation for us in Fig. 8 is that the BV using variable-level octree pruning based on the locations of 0 wavelet coefficients has the worst performance, even being outperformed by the fixed-level pruning. In fact, the fixed-level pruning performs much better than we initially expected, often being close to the rate-distortion optimal pruning. The reason for the poor R-D performance of the variable-level BV based on 0 wavelet coefficients is probably due to the fact that, for the kinds of point clouds that we tested on, the 0 wavelet coefficients are usually only found very close to the voxel level. This means that many of the original occupancy codes of the octree need to be transmitted, since the octree pruning does not go very deep into the tree, and these occupancy codes consume by far the largest proportion of the total geometry bitrate. The R-D performance of this variation of the BV method can be somewhat improved by considering not only wavelet coefficients that have a value of 0 after quantization, but also other "small" wavelet coefficients (within some threshold), which is equivalent to applying a deadzone to the uniform scalar quantizer that is used to quantize the wavelet coefficients, as explained in Section II-D. However, we do not show these results in Fig. 8, because the performance gain is not significant enough to warrant ignoring the fact that with such a deadzone applied, the reconstructed SDF values are no longer guaranteed to be within +/- $\Delta/2$ of the original SDF values, where Δ is the quantization stepsize (see the explanation in Section II-C), and so unexpected artifacts can occur.

The variable-level BV pruning that is based on considering estimated geometry reconstruction error only, without considering the estimated bitrate changes if an octree block is pruned or not, has a rate-distortion performance that is surprisingly close to the performance of the rate-distortion optimal pruning in Fig. 8. Here the distortion estimation that is used within an octree block is the maximum of the two one-way sums of squared errors between the reconstructed voxels in that block and their corresponding nearest neighbors in the same block in the original point cloud. In Fig. 8, the different points on the corresponding R-D curves represent the different squared error value thresholds as defined in the legend ("squared error thresh"), where the largest error threshold (500) produces the lowest R-D point. It is currently not clear why the performance of this pruning method is so close to the rate-distortion optimal pruning, but again, it might be related to inaccuracies in the rate/distortion estimation during the rate-distortion optimal pruning. For example, the transmitted wavelet coefficients, which, in the R-D optimal pruning case consume most of the geometry bitrate at most of the rate points, are encoded at the end (after pruning) by using rANS entropy coding per octree level, and only encoding the wavelet coefficients for the *unique* octree block corners at each level, without repetitions for shared corners. However, during rate-distortion-based pruning, the bitrates for the wavelet coefficients within a block are estimated using theoretical entropy estimates, not rANS, and currently there are no discounts in bits applied to account for the wavelet coefficients that are on shared octree corners, since we process one octree block at a time; so the overall bits estimate for the wavelet coefficients may be too high.

More generally, Fig. 8 demonstrates that the proposed BV method, when using either the rate-distortion optimal BV pruning, or the BV pruning based on estimated distortion only, or in certain cases even the fixed-level BV pruning, combined with the recursive subdivision voxel reconstruction method, outperforms the MPEG Trisoup method at most, if not all, bitrates. Both BV and Trisoup are able to achieve a very good geometry reconstruction quality (around 69 dB) at the very small bitrate of approximately 0.2 bits per input voxel, or at an even smaller bitrate than that for BV.

In addition to the better rate-distortion performance over Trisoup, the proposed Bézier Volumes method also has

Fig. 9. Geometry (shape) reconstructions obtained with BV when using *fixed-level* pruning and *recursive subdivision* voxel reconstruction. (Top leftmost) Original *boxer_8i_vox10* point cloud; (Top to bottom, left to right) Pruning levels: 3, 4, 5, 6, 7, 8. Note that a *pruning level* here means that all the descendants of the octree blocks at that level are pruned off, so that the blocks at the pruning level become leaves (or BVs). Also note that no color compression has been applied to the point clouds shown in this figure; the colors that are shown here have simply been mapped from the original point cloud by using a nearest-neighbor voxel matching between the reconstructed geometry and the original geometry.

conceptual advantages. Because the BV method represents and encodes the surface *implicitly*, by using tri-linear B-spline wavelet basis functions, this enables a naturally continuous, multiresolution, and "zoomable" surface representation, which is not the case for Trisoup. Even though the use of triangles for an explicit surface representation in Trisoup is convenient for modern graphics hardware, which is optimized to work with triangles, such a surface representation does not easily lend itself to multiresolution processing. We believe that such properties as mentioned above will be highly valuable in future applications of point clouds.

In terms of computational complexity, Tables III and IV show that for both the encoder and decoder, the proposed BV method has the same estimated *worst-case* complexity as the MPEG Trisoup approach, $O(N \log N)$.[11] However, in practice,

[11]Note that in Tables III and IV, we present the complexity estimates for the *most complex* version of the proposed BV method, which uses the rate-distortion optimal pruning. This version of the method usually produces the best rate-distortion results, as shown in Fig. 7 and Fig. 8. However, Fig. 8 also shows that some of the other, less complex versions of our algorithm often still outperform Trisoup, or at least achieve similar rate-distortion performance.

TABLE III
ENCODER COMPUTATIONAL COMPLEXITY

BV key steps	BV complexity (worst-case)	Trisoup key steps	Trisoup complexity (worst-case)
Construct octree with N occupied voxels at finest level	$O(N \log N)$	Construct octree with N occupied voxels at finest level	$O(N \log N)$
Find unique corners of all occupied octree blocks	$O(N \log N)$	Find unique edges of all occupied octree blocks	$O(N \log N)$
Compute approx. SDF (ctrl. pts.) for all unique corners	$O(N)$	Find octree block edge intersections (vertices)	$O(N)$
Compute wavelet coefficients (incl. SDF reconstruction)	$O(N)$		
Prune octree and wavelet coefficient tree (R-D optimal)	$O(N \log N)$		
Overall BV complexity (worst-case)	$\boldsymbol{O(N \log N)}$	**Overall Trisoup complexity (worst-case)**	$\boldsymbol{O(N \log N)}$

TABLE IV
DECODER COMPUTATIONAL COMPLEXITY

BV key steps	BV complexity (worst-case)	Trisoup key steps	Trisoup complexity (worst-case)
Reconstruct octree	$O(N)$	Reconstruct octree	$O(N)$
Find unique corners of all occupied octree blocks	$O(N \log N)$	Find unique edges of all occupied octree blocks	$O(N \log N)$
Reconstruct SDF	$O(N)$	Reconstruct triangles from decoded vertices	$O(N)$
Reconstruct N vox. from recon. SDF (recursive subdiv.)	$O(N)$	Reconstruct N vox. from recon. triangles (rasterization)	$O(N)$
Overall BV complexity (worst-case)	$\boldsymbol{O(N \log N)}$	**Overall Trisoup complexity (worst-case)**	$\boldsymbol{O(N \log N)}$

usually the proposed BV method (as well as Trisoup) can be made more efficient. For example, finding the unique corners of occupied octree blocks could be done in $O(N)$ expected time if a hash table is used instead of sorting, but a hash table is not very parallelizable and needs to be stored. Most of the proposed BV algorithm is highly amenable to an implementation using parallel processing techniques, because often the same operation needs to be repeated many times, e.g., on all the occupied octree blocks or unique corners at each octree level. Our current BV implementation is written in MATLAB, using highly vectorized code that is well-suited to parallelization on underlying hardware, whether multi-core on the CPU, or the GPU. Even without distributing the code across multiple cores, or transferring it to a GPU, the most computation-hungry parts of the algorithm already complete very quickly. For example, for a 10-level octree, for the point clouds in Table I, the computation of the SDF control points at almost all the octree levels takes less than 1 s, and at most takes several seconds. For the computation of the wavelet coefficients (with quantization and SDF reconstruction in-loop), the times taken are similarly small. At the decoder, most of the BV operations have linear computational complexity in the worst case (as shown in Table IV), and also complete in a few seconds. Some parts of the algorithm have been tested in a GPU implementation. For example, constructing an octree with around 1 M voxels at the finest octree level takes approximately 1 ms on an NVIDIA GeForce GTX 1080. For a similar-sized point cloud, in a single-threaded CPU (4th Gen. Intel Core i7) implementation in C/C++, the BV decoder finishes in around 140-150 ms (from decoding the input bytes to reconstructing the voxels from the reconstructed SDF), which is close to real-time. A multi-threaded GPU implementation is expected to run 2-3 times faster.

IV. CONCLUSION

Volumetric media are just that: volumetric — that is, defined as a signal on a set of points in 3D. To date, the methods for processing volumetric media have been based on points or surfaces. In Parts I and II of this paper, we propose the first approach for processing both the geometry and attributes of volumetric media in a natively volumetric way: using volumetric functions. Volumetric functions are functions defined everywhere in space, not just on the points of a point cloud. Volumetric functions, especially if they are continuous, are well-suited to representing not only an original set of points in a point cloud and their attributes, but also nearby points and their attributes. Just as regression fits continuous functions to a set of points and their values, and can therefore be used to interpolate the points and their values, volumetric functions can be used to interpolate the locations of points in a point cloud and their attributes. Especially if they are continuous, this leads to, for example, infinitely zoomable geometry without holes, as well as spatially continuous attributes. This is true regardless of the compression that may be applied to the volumetric functions, provided the compression preserves the continuity of the signal.

We choose to represent volumetric functions in a B-spline basis. Though other choices could be made, the B-spline basis maps naturally to a multiresolution, wavelet framework. Detail is thus easily preserved where necessary, and smooth regions are processed efficiently. Our multiresolution approach fits perfectly with the octree approach for geometry processing. In our approach, the octree can be considered a *non-zero tree*, which is in some sense the complement of the zero-tree used in image processing [15], in that it compactly codes the locations of non-zero (rather than zero) wavelet coefficients of the volumetric functions. When the octree is pruned, the pruned-off subtrees correspond to zero-trees, in that they compactly encode the locations of zero wavelet coefficients.

Beyond the idea of the volumetric functions for point clouds and the mechanics of coding them, our paper introduces two related, but distinct ideas. The first (in Part I) is that one of the most successful and practical transforms previously used for point cloud attribute compression, namely RAHT, can be

regarded as a continuous B-spline wavelet transform of order $p = 1$, and thereby can be generalized to higher orders, which we relate to Bézier Volumes, for higher performance. The second (in Part II) is that geometry can be compressed in its implicit representation as a level set of a volumetric function.

Experimental results confirm, for both geometry and attribute compression of point clouds, that the volumetric approach is superior to previous approaches.

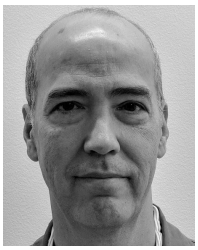
ACKNOWLEDGMENT

The authors would like to thank their colleagues at 8i, Charles Loop, Robert Higgs, and Gianluca Cernigliaro, for inspiring discussions leading to the formulation of this work, and to Bryden Frizzell for helping us achieve an almost real-time Bézier Volumes decoder implementation.

REFERENCES

[1] P. A. Chou, M. Koroteev, and M. Krivokuća, "A volumetric approach to point cloud compression, Part I: Attribute compression," *IEEE Trans. Image Process.*, vol. 29, pp. 2203–2216, 2020.
[2] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proc. 23rd Annu. ACM Conf. Comput. Graph. Interact. Techn. (SIGGRAPH)*, 1996, pp. 303–312.
[3] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones, "Adaptively sampled distance fields: A general representation of shape for computer graphics," Mitsubishi Elect. Res. Lab., Cambridge, MA, USA, Tech. Rep. TR2000-15, Dec. 2000.
[4] S. Schwarz *et al.*, "Emerging MPEG standards for point cloud compression," *IEEE J. Emerg. Topics Circuits Syst.*, vol. 9, no. 1, pp. 133–148, Mar. 2019.
[5] S. Osher and R. Fedkiw, "Signed distance functions," in *Level Set Methods Dynamic Implicit Surfaces*. New York, NY, USA: Springer, 2003.
[6] C. Loop, Q. Cai, S. Orts-Escalano, and P. A. Chou, "A closed-form Bayesian fusion equation using occupancy probabilities," in *Proc. 4th Int. Conf. 3D Vis. (3DV)*, Oct. 2016, pp. 380–388.
[7] R. A. Cohen *et al.*, "Compression of 3-D point clouds using hierarchical patch fitting," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2017, pp. 4033–4037.
[8] P. A. Chou, T. Lookabaugh, and R. M. Gray, "Optimal pruning with applications to tree-structured source coding and modeling," *IEEE Trans. Inf. Theory*, vol. 35, no. 2, pp. 299–315, Mar. 1989.
[9] S. Schwarz, G. Martin-Cocher, D. Flynn, and M. Budagavi, *Common Test Conditions For Point Cloud Compression*, document ISO/IEC JTC1/SC29/WG11 MPEG, N17766, Jul. 2018.
[10] J. Duda, "Asymmetric numeral systems: Entropy coding combining speed of Huffman coding with compression rate of arithmetic coding," Jan. 2014, *arXiv:1311.2540v2*. [Online]. Available: https://arxiv.org/abs/1311.2540v2
[11] *7-zip*. Accessed: Aug. 1, 2019. [Online]. Available: https://7-zip.org
[12] *7z format*. Accessed: Aug. 1, 2019. [Online]. Available: https://7-zip.org/7z.html
[13] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337–343, May 1977.
[14] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inf. Theory*, vol. 24, no. 5, pp. 530–536, Sep. 1978.
[15] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Signal Process.*, vol. 41, no. 12, pp. 3445–3462, Dec. 1993.

Maja Krivokuća received the Bachelor of Engineering degree (Hons.) in computer systems engineering and the Ph.D. degree from the University of Auckland, New Zealand, in 2010 and 2015, respectively. Her Ph.D. research was on developing a new and improved algorithm for the progressive compression of 3D mesh geometry, using redundant dictionaries and sparse representation techniques. Since then, she has worked for Mitsubishi Electric Research Laboratories, Cambridge, Massachusetts, USA, researching new methods for compressing large-scale 3D point clouds obtained from Mobile Mapping Systems, and for 8i, Wellington, New Zealand, researching and developing new shape compression algorithms for 3D point clouds in virtual/augmented reality applications. She has also been actively involved in contributing to the emerging MPEG Point Cloud Compression standards, since January 2017. She is currently at Inria, Rennes, France, working on research related to light field technology as part of the SIROCCO research team and the Computational Light Fields Imaging project. Her main research interests are in the areas of data compression and information theory, and in signal and image processing.



Philip A. Chou (F'81) received the B.S.E. degree from Princeton University, the M.S. degree from the University of California, Berkeley, and the Ph.D. degree from Stanford University. He has been on the research staff at AT&T Bell Laboratories, the Xerox Palo Alto Research Center, Microsoft, and Google. He has worked for startups Telesensory Systems, Speech Plus, VXtreme, and 8i. He has been on the affiliate faculty at Stanford University, the University of Washington, and the Chinese University of Hong Kong. He has served on IEEE Fellow evaluation committees for the IEEE Signal Processing and Computer societies. He has been on the organizing committees of numerous conferences and workshops, including ICASSP and ICIP. He has served on the Board of Governors and Technical Committees for the IEEE Signal Processing Society. He has received the best paper awards from the IEEE TRANSACTIONS ON SIGNAL PROCESSING and the IEEE TRANSACTIONS ON MULTIMEDIA, and from several conferences. He is co-editor of a book on *Multimedia Communication*. He has been an Associate and/or Guest Editor of the IEEE TRANSACTIONS ON INFORMATION THEORY, the IEEE TRANSACTIONS ON IMAGE PROCESSING, the IEEE TRANSACTIONS ON MULTIMEDIA, and the IEEE SIGNAL PROCESSING MAGAZINE.

Maxim Koroteev graduated with the degree (Hons.) in applied mathematics and physics from Moscow Institute of Physics and Technology (MIPT). After getting his Ph.D., he worked for around fifteen years in various academic and industrial research institutions. In 2016, he joined 8i as a Researcher in computer vision. He then worked as a Data Scientist for a company designing mobile autonomous vehicles. He is currently with Imatrex Inc. to work on image reconstruction for a new generation CT scanner. His interests vary from statistical physics and bioinformatics to computer vision and robotics.