Fractional Super-Resolution of Voxelized Point Clouds

Tomás M. Borges, Diogo C. Garcia, *Senior Member, IEEE,* and Ricardo L. de Queiroz, *Fellow, IEEE*

***Abstract*—We present a method to super-resolve voxelized point clouds downsampled by a fractional factor, using lookup-tables (LUT) constructed from self-similarities from their own downsampled neighborhoods. The proposed method was developed to densify and to increase the precision of voxelized point clouds, and can be used, for example, as improve compression and rendering. We super-resolve the geometry, but we also interpolate texture by averaging colors from adjacent neighbors, for completeness. Our technique, as we understand, is the first specifically developed for intra-frame super-resolution of voxelized point clouds, for arbitrary resampling scale factors. We present extensive test results over different point clouds, showing the effectiveness of the proposed approach against baseline methods.**

***Index Terms*—Point clouds, super-resolution, resampling.**

I. Introduction

AMONGST 3D representation alternatives, point-cloud (PC) imaging gained popularity due to its relatively low-complexity and high-efficiency in capturing, encoding, and rendering of 3D models. A PC is a list of points in the 3D space, each with spatial coordinates (x, y, z) and attributes like colors, normals, reflectances, etc. For a single-color (RGB) attribute per point, a PC is defined by its geometry V and its color C sets:

$$V = \{\mathbf{v}(k)\}, \text{ with } \mathbf{v}(k) = (x_k, y_k, z_k), \text{ and}$$
$$C = \{\mathbf{c}(k)\}, \text{ with } \mathbf{c}(k) = (R_k, G_k, B_k).$$

Occupied points can have any (x, y, z) real values, however this makes PCs cumbersome to process and encode. Over the past years, The Moving Picture Expert Group (MPEG) has put an effort to develop efficient point cloud compression (PCC) techniques and standardization [1], [2]. One of the requirements imposed by current MPEG PCC standards is the use of voxelized PCs, in which points are made to lie in an integer grid such that their geometry is quantized. Input coordinates are transformed such that all points lie within a bounding cube $[0, 2^d)^3$, for some non-negative integer parameter d. Voxels (volume elements) represent the center of any of the unit cubes $[i - 0.5, i + 0.5) \times [j - 0.5, j + 0.5) \times [k - 0.5, k + 0.5)$, for i, j, k integers between 0 and $2^d - 1$. Duplicate points inside a voxel are usually merged and their attributes are averaged.

Work partially supported by CNPq under grants 88887.600000/2021-00 and 301647/2018-6.

T. M. Borges is with the Electrical Engineering Department at Universidade de Brasilia, Brasilia, Brazil, e-mail: tomas@divp.org.

D. C. Garcia, is with the Gama Engineering College, Universidade de Brasilia, Brasilia, Brazil, e-mail: diogogarcia@unb.br.

R. L. de Queiroz is with the Computer Science Department at Universidade de Brasilia, Brasilia, Brazil, e-mail: queiroz@ieee.org.

The resampling of PCs, i.e., the changing in the number of points of a PC, can be approached in two ways. The first decimates/populates points in the original set without changing the voxel resolution. We refer to this as *set resampling*. The second approach changes the number of points by revoxelizing the PC using a different size of voxel, i.e., changing the volumetric resolution. We refer to it as *grid resampling*. Applications that require the downsampling of PCs, such as lossy-compression and rendering, can profit from methods that create high-resolution (HR) PCs from low-resolution (LR) versions, i.e., super-resolution (SR). In this paper, we propose an SR method that takes a grid-downsampled LR voxelized PC as input and combines the restrictions imposed by the grid, its density, and self-similarities to output a grid-super-resolved version of that input. One popular way to organize the points is to use *octrees* [3]. For example, MPEG's Geometry-based PCC (G-PCC) uses pruned octrees combined with other techniques for lossy-compression [1], [2], [4], [5]. PC SR methods involving octrees have not been well-explored and may be useful in rendering and compression, as well as in other typical SR applications, such as reduced PC capture resolution and enhancement of inadequately-captured PCs.

Although many approaches exist, often they cannot be directly compared because of different number of inputs, different LR versions, or even the use of extra information. In optimization-based PC SR, a cost function is defined, then new points are added to minimize this function. Alexa *et al.* [6] proposed constructing a Voronoi diagram on the 3D surface, then inserting points in the vertices to minimize a moving least squares (MLS) cost function. Other works were also developed using the MLS cost function [7], [8], but tend to over-smooth the geometry. Huang *et al.* [9] introduced an edge-aware solution to mitigate the over-smoothness of prior methods, relying on the accuracy of normals and on a thorough parameter tuning. Hamdi-Cherif *et al.* [10] combined local descriptors by their similarities for PC SR, however this approach required several PCs (multiple scans), normals calculations, and assumed surface smoothness. Dinesh *et al.* [11], [12] used Delaunay triangulation in the LR PC and optimize an L_1-norm graph-total-variation (GTV) cost function for neighborhood surface normals. It promotes piecewise smoothness in reconstructed 2D surfaces, under the constraint that the LR coordinates are preserved. The optimization-based SR methods we have found were developed for static non-voxelized PC, and are considered set upsamplings. Dinesh *et al.* did use one voxelized PC in their tests [12], but it was unclear if the SR version was still voxelized.

Deep learning was used for PC SR with the introduction of

PU-NET by Yu *et al.* [13], which learns multi-scale features by downsampling the input and expands the point set via multi-branch multilayer perceptrons (MLP). Yu *et al.* also proposed EC-NET [14], an edge-aware network for point consolidation, alas, it requires a very expensive edge-notation for training. Wang *et al.* [15] proposed a progressive network, 3PU, to suppress noise and to preserve details in the upsampling geometry. It is computationally expensive, though, and requires a lot of training data. PU-GAN [16] was designed to obtain more uniformly distributed SR results, with its major contribution and performance gains coming from the discriminator part. Graph convolutional networks (GCN) were used for PC SR by Wu *et al.* [17], and by Qian *et al.* [18]. PUGeo-Net [19] proposes to perform the upsampling by learning the first and second fundamental forms to represent the local geometry, although normals are required. Nonetheless, all these networks require retraining when different upsampling scales are required, making them cumbersome for dynamic applications. Recently, an independent work by Ye *et al.* [20] proposed Meta-PU, a network that supports upsamplings for arbitrary scales without the need of retraining, using meta-learning to predict the weights of the network and dynamically change behavior for each scale factor. The type of upsampling and the PC standards needed for data-driven methods could change with retraining. Yet, all the works we have found were trained with and promoted set upsampling for static non-voxelized PC. LiDAR PCs were also tested [16], [18], [19].

Previous methods did not consider voxelization, so it is not guaranteed they would work with technologies that have such requirement, e.g., G-PCC. Octree-based PC SR, on the other hand, already copes with those requirements. Such methods usually exploit spatial correlation from the tree to expand pruned branches, thus improving precision and adding new points to the geometry (grid upsampling). Garcia *et al.* developed two SR methods for dynamic PCs based on statistics gathered in previous frames of the sequence: SR by example and SR by neighborhood inheritance [21]. The first explores similarities between time-adjacent frames to predict voxels at higher levels of the octree [22]. The LR frame is super-resolved using the similarities from the previous frame at full-resolution. The second creates a dictionary of child nodes based on the neighborhood configuration from previous full-resolution frames. Then, the neighborhood from each occupied voxel is used to estimate its child nodes.

The proposed method is an expansion of the neighborhood inheritance method [21], where we removed the need for extra PCs during dictionary creation (intra SR), and we developed a better understanding of fractional resampling, which allowed for arbitrary scale factors and not only powers of 2.

II. Point Cloud Resampling

A. Downsampling

In set downsampling, points are usually decimated using some distance-based criterion, like Poisson disk sampling [23]. This approach reduces the PC density, since it is not relative to voxel size, and creates "holes" and "isolated points", reducing spatial correlation and, thus, affecting coding efficiency.

Fig. 1: Downsampling approaches. Original PC (857,966 voxels), set downsampling (64,176 voxels $D1_{PSNR}$ = 61.5dB), and grid downsampling (62,130 voxels, $D1_{PSNR}$ = 58.4dB).

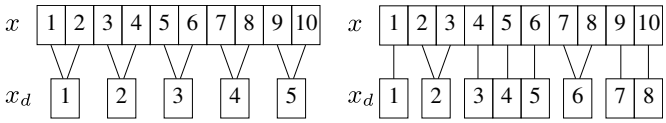
In grid downsampling, points are decimated through voxelization. While the resolution is lowered, the density increases, compared to the original PC, thus rendering a "blocky" geometry. The remaining points in the LR PC cannot be rescaled to its original resolution without error. Figure 1 illustrates PC downsampling using both approaches to arrive at approximately the same number of points. In order to compare both PCs, the grid-downsampled one was expanded and rendered with larger voxels. We also show distortion numbers using the $D1_{PSNR}$ metric, described in Sec. IV.

Grid downsampling can be achieved by dividing the geometry V by a scale factor $s > 1$, and rounding the results to snap them to yet another integer grid. Duplicate points are merged, and their textures are averaged. Thus, we achieve a downsampled geometry V_d using:

$$V_d = \texttt{unique}\left(\texttt{round}\left(\frac{V}{s}\right)\right), \tag{1}$$

where $\texttt{unique}(X)$ is the function that only returns the unique vectors in the set X, and $\texttt{round}(\cdot)$ is the function that rounds the components of a vector to the nearest integer. The consolidation of duplicate position information is similar to the voxelization process.

V_d and V form a hierarchical tree structure, such that the former represents parent nodes, and the latter represents child nodes. This is illustrated in 1D in Fig. 2, where we can see that when s is an integer, the number of child nodes is equal for all coordinates in x. This downsampling process is regular as all parent nodes have the same number of children. However, when s is not an integer, the number of child nodes varies depending the parent node's coordinate. For $1 < s < 2$, there are parent coordinates with only one child (uniparous) and others with two children (multiparous).



(a) $s = 2$ (b) $s = 1.25$

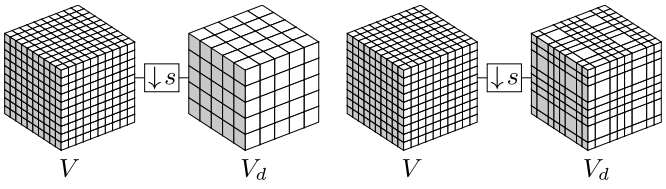
Fig. 2: Illustration of the downsampling process over a fully-occupied single-axis x. It is also possible to see a hierarchical tree configuration with x_d as parent nodes of x.

In 3D, regular downsampling (integer values of s) translates to every group of voxels in a $s \times s \times s$ cube in V being reduced

to just one voxel in V_d, as depicted in Fig. 3(a). For this case, parent nodes in V_d have s^3 children. This represents a pruning of the original octree structure. When $s = 2^n$, $n = 1, 2, 3, \ldots$, the pruning occurs exactly[1] at level $d - n$. When $s \in \mathbb{Q}$, parent nodes in V_d have *up to* s^3 children. For example, when $1 < s \leq 2$ parent nodes may have 1, 2, 4 or 8 children, depending on each parent node's coordinate value, as depicted in Fig. 3(b). If a parent node has x multiparous, and y and z uniparous, it, thus, has $2 \cdot 1 \cdot 1 = 2$ child nodes, and at least one of them must have been occupied in V. The number of possible children for a given parent can be generalized as

$$i_{\max} = (\lceil s \rceil - 1)^u \lceil s \rceil^m, \tag{2}$$

where u is the number of uniparous coordinates, m the number of multiparous coordinates (here we extend the meaning of uniparous to indicate parents with fewer children than the multiparous ones), and $\lceil\ \rceil$ means the ceiling operator. Although a non-integer value of s produces an irregular voxel grid, there is a pattern on such grid when s is rational in the form p/q, for $p > q$. We call fractional resampling the use of a non-integer value of s to perform down- or upsampling.



(a) $s = 2$ (b) $s = 1.25$

Fig. 3: Downsampling in the voxel grid. For a fractional value of s in (b) the different parenthood relationships are manifest.

B. Upsampling

We define PC grid upsampling as the inverse of the just-defined grid downsampling process. The space of the bounding cube containing the voxelized PC is again re-quantized, this time to increase spatial resolution. It can be done by the simple expansion of the downsampled geometry,

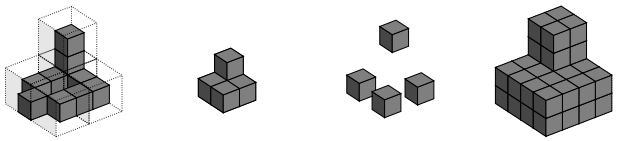
$$V_e = \texttt{round}\left(V_d \cdot s\right). \tag{3}$$

However, it yields a very sparse PC. In order to densify the upsampled version, we need to interpolate the missing points. The baseline technique for completing the missing points is the nearest-neighbor interpolation (NNI), which sets as occupied all children from the parent nodes in V_d. The texture upsampling for the NNI usually follows the same idea used for the geometry: the colors from parent nodes are just replicated to their corresponding children. Figure 4 illustrates what happens to a geometry after being downsampled, expanded, and, finally, upsampled using NNI.

To express the NNI, we have that all child nodes $\{\mathbf{v}_u(i)\}$ from a parent voxel $\mathbf{v}_d(k)$ satisfy,

$$\mathbf{v}_d(k) = \texttt{round}(\mathbf{v}_u(i)/s), \tag{4}$$

[1]The term *exactly* is somewhat relaxed here because of the use of `round`. To get the strictly exact pruning equivalence, Eq. (1) should be defined using the `floor` function instead.



(a) V (b) V_d (c) V_e (d) V_u

Fig. 4: Illustration of the NNI upsampling for $s = 2$: (a) child nodes (V) surrounded by its parent nodes; (b) parent nodes (V_d); (c) expanded geometry; (d) NNI upsampling.

for every $i = 1, 2, \ldots, i_{\max}$. Inversely,

$$\mathbf{v}_u(i) = \texttt{round}(s \cdot \mathbf{v}_d(k)) + \epsilon(i), \tag{5}$$

where $\epsilon(i)$ is the rounding error. Let $\mathcal{E}(k) = \{\epsilon(i)\}$ be the set containing all the $i_{\max}$ error samples for the parent node $\mathbf{v}_d(k)$. Thus, the set containing all possible children from $\mathbf{v}_d(k)$ is

$$\mathcal{V}_u(k) = \texttt{round}(s \cdot \mathbf{v}_d(k)) + \mathcal{E}(k). \tag{6}$$

Therefore, the geometry from the NNI upsampling is

$$V_u = \texttt{unique}\left(\bigcup_{k=1}^{K} \mathcal{V}_u(k)\right). \tag{7}$$

The colors for $\mathcal{V}_u(k)$ are $\mathcal{C}_u(k) = \{\mathbf{c}_u(i)\},\ i = 1, 2, \ldots, i_{\max}$, such that, $\mathbf{c}_u(i) = \mathbf{c}_d(k)$, and

$$C_u = \bigcup_{k=1}^{K} \mathcal{C}_u(k). \tag{8}$$

In order to improve the quality of the upsampled geometry V_u, further processing can be applied to smooth both its geometry and its texture. Laplacian smoothing (LS) [24], for example, is used to smooth meshes and can be adapted to work on PCs. Briefly, with LS, for each occupied voxel in a given input PC, its $3 \times 3 \times 3$ neighborhood is analyzed, and the average voxel position is computed. This average is rounded to the integer grid, and it becomes the center voxel's new position. Once the geometry smoothing is complete, output voxels receive attributes from their nearest neighbors in the input PC. If there is an input voxel at the same position, the output voxel inherits its attributes. Otherwise, the average attribute of the nearest neighbors is used. LS can improve NNI results by reducing the aliasing caused by coarse interpolation.

III. Intra-frame super-resolution of voxelized point clouds

A. Proposed SR geometry method

The idea of using a dictionary of child nodes was first used by Garcia *et al.* [21]. It was devised for the inter-frame case to super-resolve LR frames downsampled by a scale factor of $s = 2^n$. In their research, a dictionary, or look-up-table (LUT), relating child nodes to neighborhood configuration was created from an HR reference frame. SR was achieved by matching LR voxels' neighborhood configuration with the correspondent child occupancy given by the previously built LUT.

We developed our method using a similar process, i.e., relating child nodes with neighborhood configuration. However, the dictionary is now built using the very same PC

we want to super-resolve. Furthermore, we introduce fractional downsampling. Hence, an additional step for geometry classification was developed to cope with irregularities of such downsampling, preserving the parent-child relationship in the upsampling scheme. Our method's general scheme is illustrated in Fig. 5. The steps in gray boxes are as follows.

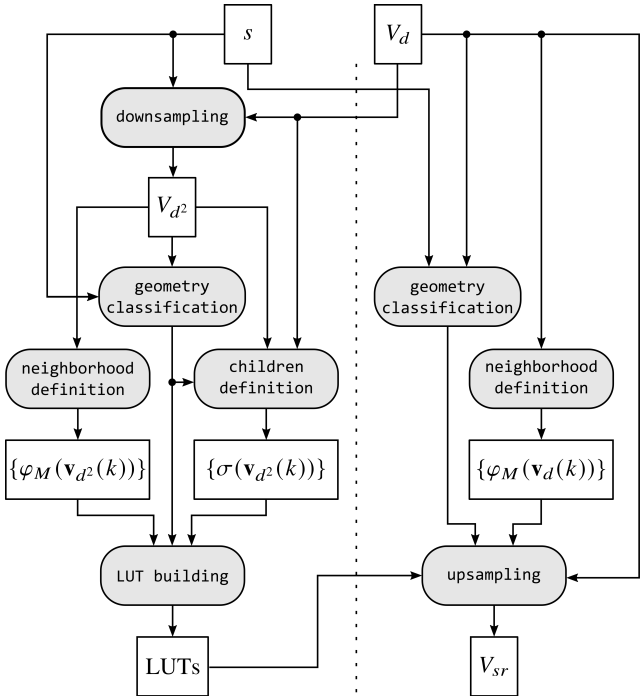


Fig. 5: General scheme of the proposed method.

Neighborhood definition: For each voxel, we verify its surroundings to define its neighborhood occupancy. Let $\varphi_M(\mathbf{v}(k))$ be a (M^3-1)-binary number indicating the occupancy of neighbor voxels inside an $M \times M \times M$ cube around voxel $\mathbf{v}(k)$. The smallest neighborhood, $M=3$, leads to $3^3-1=26$ neighbors (adjacent voxels).

Children definition: Let the child occupancy configuration of parent voxel $\mathbf{v}_d(k)$ be defined as $\sigma(\mathbf{v}_d(k))$, a $\lceil s \rceil^3$-binary number indicating which of the possible children of $\mathbf{v}_d(k)$, i.e., $\mathcal{V}_u(k)$ in our notation, are indeed occupied. Unlike [21], we take the input geometry V_d and perform yet another downsampling using potentially the same scale factor[2] s, as the one used to arrive in V_d, to generate the parent geometry V_{d^2}. In this way, we can define the child occupancy configuration for each parent voxel $\sigma(\mathbf{v}_{d^2}(k))$. In Fig. 6, we illustrate the mapping of neighborhood and child occupancies.

Geometry classification: We divide downsampled voxels into classes, depending on the position and number of possible children, i.e., which values of (x,y,z) are uniparous and which are multiparous. By doing so, we avoid dealing with grid irregularities. Figure 7 illustrates the eight possible classes for the geometry, for a scale factor $1 < s \leq 2$. This module is used whenever we need to estimate child nodes in a fractional grid. It is used for defining children's occupancies, building the LUT (allowing for the creation of one LUT for each class), and in the upsampling module.

[2]The assumption of a known s is made so we can compare the super-resolved PC with its original version. This is true for compression post-processing applications. When the original value of s is unknown, the method can still be used, but without being able to arrive at the original resolution.

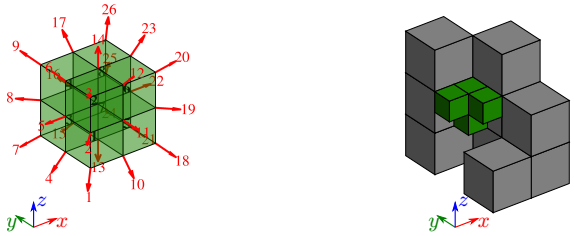


Fig. 6: On the left, the order of bits representing neighbors (red) and children (black). On the right, an example of neighborhood where $\varphi_3 = \texttt{000 111 011 111 00 001 000 000 000}$ and $\sigma = \texttt{1110 1000}$.

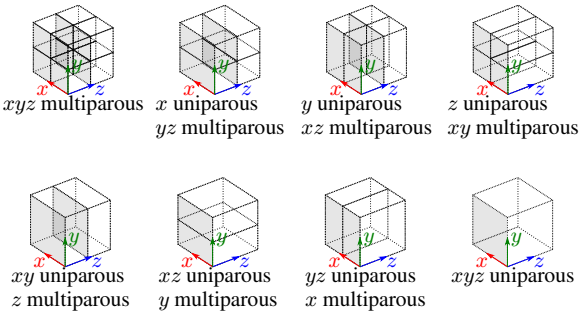


Fig. 7: The 8 classes of geometry classification for $1 < s \leq 2$.

LUT building: For each voxel in V_{d^2} from each class, we can relate its neighborhood $\varphi_M(\mathbf{v}_{d^2}(k))$ with its occupied children $\sigma(\mathbf{v}_{d^2}(k))$. Thus, we can create one LUT for each class that will tell us how to upsample a parent voxel depending on its neighborhood. Each LUT should pair all the 2^{M^3-1} possible neighborhood configurations with an output child occupancy. The m-th entry of each LUT is created by estimating the most likely child occupancy for $\varphi_M(m)$,

$$\bar{\sigma}(m) = E\{\sigma(\mathbf{v}_{d^2}(k)) \mid \varphi_M(m)\}, \tag{9}$$

i.e., the expected value (bitwise mean) of all child occupancies sharing the same neighborhood configuration. Neighborhood configurations not present in the input data are associated with fully-occupied child nodes.

In order to save memory, we can store only the N_f found neighborhoods, and infer the missing entries as having fully-occupied child states in the upsampling module. In this way, each LUT is an N_f-by-2 array, pairing a neighborhood configuration in the first column with its correspondent child occupancy in the second column. Creating and storing LUTs for PCs with millions of points is somewhat memory-intensive. Because of that, and due to empirical findings, some constraints were imposed. For a symmetric neighborhood, M should be odd. As M increases by a single step, from 3 to 5, the possible neighborhood configurations go from 2^{26} to 2^{124}. It is impractical to use large values of M because it takes more computational effort to find a larger neighborhood. Furthermore, the entries of such large dictionary become overly specific, and the output geometry approaches the NNI

upsampling. For this reason, we decided to fix $M = 3$, such that, whenever $\varphi(k)$ is mentioned, it is implicit that a neighborhood size 3, $\varphi_3(k)$, is considered.

Due to memory and software limitations, we were limited to $s \leq 3$. Moreover, as s increases, the number of meaningful entries in the dictionary decreases, since there is not much information in the lower levels of the geometry. As s increases, the probability of making a right guess about the estimated children is dramatically reduced, from $1/255$ for $1 < s \leq 2$, to $1/19.982$ for $2 < s \leq 3$. In other words, the preservation of self-similarities is diminished with the increase of s. Thus, we decided to constrain the values of s, $\{s \in \mathbb{Q} \mid 1 < s \leq 2\}$. Inside this interval, we can profit from partial downsampling and super-resolve a full octree level. If $s > 2$ is required, the proposed SR method can still be used in a cascading manner. For example, by performing $t = \lceil \log_2(s) \rceil$ nested SRs with a new scale factor $s' \approx \sqrt[t]{s}$. Empirically, we have noticed that consecutive upsamplings work better than a single one for $s > 2$, although slower.

As a means of data augmentation, we applied incremental translations to the input frame to increase the LUT population. Since $1 < s \leq 2$, shifts of ± 1 in each axis are sufficient to change the result of Eq. (1), and the parent geometry classification. Other transformations, such as rotation or scaling, are left to future work.

Upsampling: The upsampling module first performs the NNI in the input PC, in order to locate all possible children of each parent. Then, it removes extra points using the LUTs. We label each voxel in V_d and use the correspondent LUT, for each class, to consult which of the voxels from the NNI should be removed, based on $\varphi(\mathbf{v}_d(k))$. We super-resolve from V_d to a higher resolution V_{sr} by carving the NNI geometry. The set of super-resolved children from a single parent voxel $\mathbf{v}_d(k)$ is

$$\mathcal{V}_{sr}(k) = \mathcal{V}_u(k \mid \bar{\sigma}(\mathbf{v}_{d,j}(k))), \tag{10}$$

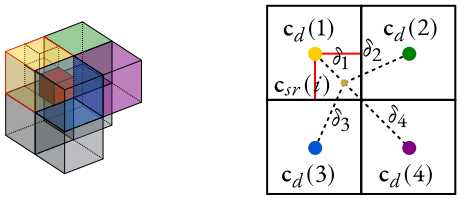
where $\bar{\sigma}(\mathbf{v}_{d,j}(k))$, found by consulting LUT_j for $\varphi(\mathbf{v}_{d,j}(k))$, indicates that a given neighborhood configuration determines which of the possible children of $\mathbf{v}_d(k)$ should be set as occupied, and $0 \leq j \leq 7$ indicates the geometry class (Fig. 7). The super-resolved geometry is the union of all $\mathcal{V}_{sr}(k)$ as

$$V_{sr} = \bigcup_{k=1}^{K} \mathcal{V}_{sr}(k). \tag{11}$$

B. Color interpolation

In order to find the texture for the upsampled geometry, the usual approach is to first represent the LR geometry at the same scale as the SR geometry, then to interpolate the colors for the SR voxels using a distance-based weighted average of the LR colors, taken over a $3 \times 3 \times 3$ neighborhood.

We can improve the interpolation method by borrowing the color prediction scheme used in G-PCC. In the transform domain prediction of region-adaptive hierarchical transform (RAHT) coefficients [1], [5], the estimated color for each occupied child is the average of the parent's color, with the colors of the "uncles" that share an edge with that child, as illustrated in Fig. 8. The average is weighted by the inverse distance between each parent and the current child being estimated. We refer to this method as the weighted average of adjacent neighbors (WAAN). A variable weight ζ, dependent of s, is introduced in the WAAN to take into account the idea that the parent color should be more important as the scale factor decreases.



(a) (b)

Fig. 8: Illustration of the neighbors used in the WAAN calculation. (a) For the highlighted child, only "uncles" sharing an edge with it are considered. (b) The distances δ_ℓ, from the child node to its "uncles".

Thus, $\mathcal{C}_{sr}(k) = \{\mathbf{c}_{sr}(i)\}$ is the set containing the respective colors of $\mathcal{V}_{sr}(k)$ of a given parent $\mathbf{v}_d(k)$, such that:

$$\mathbf{c}_{sr}(i) = \frac{\mathbf{c}_d(k) + \zeta \sum_\ell \delta_\ell^{-1} \mathbf{c}_d(\ell)}{1 + \zeta \sum_\ell \delta_\ell^{-1}}, \tag{12}$$

where ℓ is the index of the occupied voxels in V_d sharing an edge with $\mathbf{v}_{sr}(i)$, and the δ_ℓ are their Euclidean distance (Fig. 8(b)). ζ was empirically found as $\zeta = \delta_1 s/8$. The super-resolved colors are

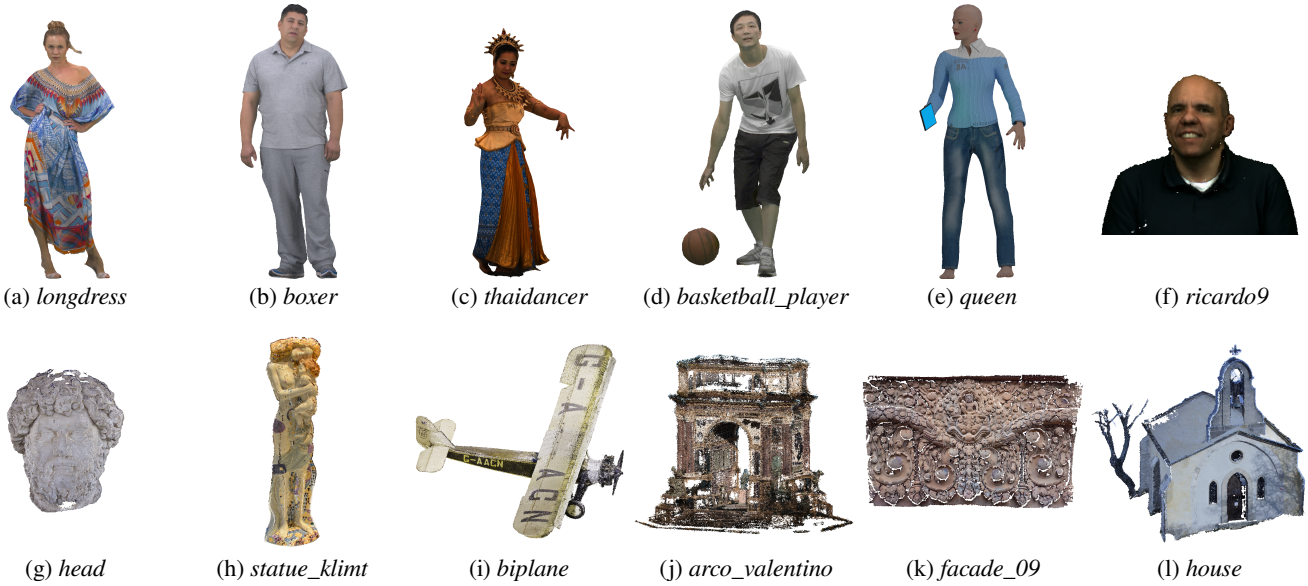
$$C_{sr} = \bigcup_{k=1}^{K} \mathcal{C}_{sr}(k). \tag{13}$$

IV. Performance Assessment and Analysis

A. Datasets and test conditions

We focused on PCs of static objects and scenes from the MPEG's G-PCC common test conditions (CTC) [25]. We set a point cap of just over 4 million voxels, due to the current implementation's memory restrictions. Table I summarizes information on the tested PCs, divided by categories in which the PCs share similar metrics. In the table, ρ_φ is a density measure: average number of occupied neighbors to any given occupied voxel. ρ_φ is divided by 26 to be a relative measure. The column "Vox." indicates whether the PC required to be revoxelized as a pre-processing step. The revoxelization was done in two cases: to reduce PCs having more than our point cap, and to increase density of extremely sparse clouds, where a downsample using $s = 2$ would decimate less than 1% of the original points. Figure 9 illustrates the chosen PCs.

ρ_φ can be used as a performance predictor of the proposed method, since we rely on similarities at different scales. Those are somewhat maintained for dense PCs, but not so much for sparse ones. We empirically found that when ρ_φ is beyond 0.3 or so, the PC has watertight projections, i.e., there is a one-to-one relationship between rendered pixels and voxels



(a) *longdress* (b) *boxer* (c) *thaidancer* (d) *basketball_player* (e) *queen* (f) *ricardo9*

(g) *head* (h) *statue_klimt* (i) *biplane* (j) *arco_valentino* (k) *facade_09* (l) *house*

Fig. 9: Representative viewpoints of some of the human figures from (a) to (f), and of the objects from (g) to (l).

TABLE I: Summary of tested PCs.

Point clouds	Vox.	Depth	# voxels	ρ_φ
(a) Group: *8i_vox10* [26]†				
longdress_vox10_1300	✗	10-bit	857,966	0.429
loot_vox10 _1200	✗	10-bit	805,285	0.428
redandblack_vox10_1550	✗	10-bit	757,691	0.433
soldier_vox10_0690	✗	10-bit	1,089,091	0.432
(b) Group: *8i_vox12* [27]‡				
boxer_viewdep_vox12	✗	12-bit	3,493,085	0.031
longdress_viewdep_vox12	✗	12-bit	3,096,122	0.027
loot_viewdep_vox12	✗	12-bit	3,017,285	0.029
redandblack_viewdep_vox12	✗	12-bit	2,770,567	0.025
soldier_viewdep_vox12	✗	12-bit	4,001,754	0.026
(c) *Thaidancer_viewdep_vox12* [27]‡	✗	12-bit	3,130,215	0.332
(d) Group: *owlii* [28]‡				
basketball_player_vox11_00000200	✗	11-bit	2,925,514	0.452
dancer_vox11_00000001	✗	11-bit	2,592,758	0.445
(e) *queen_frame_0200*‡	✗	10-bit	1,000,993	0.524
(f) Group: *MVUB* [29]†				
andrew9_0000	✗	9-bit	279,664	0.547
david9_0000	✗	9-bit	330,797	0.542
phil9_0000	✗	9-bit	370,798	0.543
ricardo9_0000	✗	9-bit	214,656	0.550
sarah9_0000	✗	9-bit	302,437	0.538
(g) *Head_00039_vox12*‡	✓	9-bit	938,112	0.532
(h) *1x1_Biplane_Combined_000*†	✓	10-bit	1,181,016	0.567
(i) *Statue_Klimt_vox12*‡	✓	10-bit	483,068	0.209
(j) *Arco_Valentino_Dense_vox12*‡	✗	12-bit	1,481,746	0.025
(k) *Facade_00009_vox20*‡	✓	11-bit	1,560,786	0.165
(l) *House_without_roof_00057_vox12*‡	✓	11-bit	3,638,139	0.247

† https://jpeg.org/plenodb/
‡ https://mpegfs.int-evry.fr/mpegcontent/

without holes[3]. We consider PCs with watertight projections to be somewhat dense. If, however, ρ_φ is below 0.3, we considered the PC to be sparser, as there will likely be holes in its projections. If $\rho_\varphi = 0$ for a given neighborhood size M, but $\rho_\varphi > 0$ for a neighborhood size $M' > M$, then it is possible to reduce an initially assumed sparse PC into a dense one at a lower resolution.

[3]These findings consider an orthographic voxel-based rendering approach, with the voxel size adaptively following the grid size for different zoom levels. We had not tested with other PC rendering approaches.

B. Self-similarities at different scales

The proposed SR method assumes that the geometry of a PC is approximately self-similar at different scales. The dictionaries are constructed using a coarser geometry and applied to recreate a finer geometry. One way to verify the preservation of similarities at different scales is to create a dictionary using the original HR PC and to compare it with the one created from the LR version.

In order to measure similarities, we calculated the Intersection over Union (IoU), i.e., the ratio of coincidence, for different scales. We only considered the set of neighborhood configurations contained in $\{\varphi(\mathbf{v}_d(k))\}$.

From Fig. 10, we see that similarity is mostly maintained over $1 < s < 1.5$, where there is a large number of uniparous parents. There is a fast decay for $1.5 < s < 2$, where multiparous parents become the majority. Higher similarity levels are found in "well-behaved" PCs, i.e., dense and with smaller noise levels, which is the case of groups (a), (c), (d), and (e). For sparse clouds, similarity is lower and usually decreases faster with the increase of s. The exception being (j) which seems to have a different behavior. However, this PC is so sparse that most of its voxels are isolated, since $\rho_\varphi = 0.025$ indicates that each voxel has on average 0.65 neighbor. Thus, its plot reflects only a handful of entries. For $2 < s < 3$, the similarity has a slower decay. Low IoU levels indicate that the method would struggle for such scale factors. Note that IoU only accounts for exact matches between dictionaries, but a partial match can still be beneficial for our purposes.

C. Evaluation framework

We focused our evaluation in the range of $1 < s \leq 2$, where we believe our method yields best results, regarding

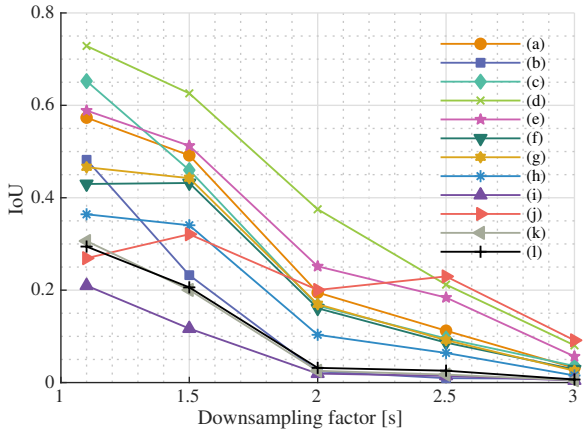


Fig. 10: IoU measurements comparing LR with HR LUTs at different scale factors.

complexity and distortion. For $s > 2$, the number of points added by the NNI is quite large, and the self-similarities are diminished, making the NNI carving less effective. We compare the proposed method (LUT) with the NNI, and also considered smoothing it using the LS technique (labeled as NNI+LS). Since the NNI can be replicated by the renderer, just by geometry expansion and using a larger cubic voxel size, we chose not to use the WAAN for its colors to better evaluate the trade-off between complexity and distortion. The alternative for smoother colors is the NNI+LS.

Comparisons with other methods were not carried out because they were developed with set downsampling in mind and would require adaptations to work with grid-downsampled LR PCs. Also, most of the deep learning methods would require a lot of retraining to cope with the different scale factors and with the real-world voxelized PCs of our test set. The methods from Garcia *et al.* [21] also cannot be used because they do not allow for fractional scale factors, nor do they work for intra-frame SR.

The following PC metrics were chosen for the assessment.

- Point-based: Point-to-point ($D1_{PSNR}$) [30], Point-to-plane ($D2_{PSNR}$) [31], and Luma end-to-end (Y-PNSR).
- Projection-based [32], [33]: PPSNR, PSSIM [34], and PVIFp [35].

Point-based metrics: symmetrically computed, first using the original PC as reference, then using the distorted super-resolved version as reference. The final value is the maximum error over the two measurements. Below is a brief description of such metrics:

- *D1*: the average squared distance between each point in the first PC and its nearest neighbor in the second one.
- *D2*: similar to the D1 metric, except that the distances are projected to the normal direction before being averaged, integrating local plane properties.
- *Y-PSNR*: RGB colors are converted to YUV_{709}, then the Y channel of each point in the first PC is compared to the one from their nearest neighbors in the second PC.

D1 and D2 are converted to PSNR values using the PC's bounding cube's diagonal length as the peak distance value.

Projection-based metrics: image metrics adapted for PCs. They are referred with a preceding "P", as in projected PSNR (PPSNR). Voxels were rendered as cubes to get the projections, and point size was set equal to 1. Although this choice of rendering may generate holes in sparse PCs, different choices could add rendering distortions to SR intrinsic artifacts. Six orthographic projection views were used (the six faces of the cube containing the PC). In this way, each visible voxel face is projected into a single pixel, hence, a depth-10 PC will have six 1024×1024 pixel projections. In order to decrease the effect of the background in the metrics, its color was set to a mid-gray value, and we only considered the rectangular region formed by the union of the foregrounds of the reference and the distorted projections, as suggested by Alexiou *et al.* [36]. The PPNSR is calculated in the RGB color space, taking the average of the PSNR for each view:

$$\text{PPSNR} = \frac{1}{6}\sum_{n=1}^{6} \text{PSNR}_n. \tag{14}$$

The calculation of PSSIM and PVIFp is similarly performed, but only considering the Y channel, using YUV_{709} conversion.

D. Results

Table II shows the average gain of the NNI+LS and the LUT approaches when compared to the NNI over all scale factors $1 < s \leq 2$. As we can see from this table, the proposed method is superior to the baseline or its smoothed counterpart for every content in almost every metric, especially when point-based geometric metrics are considered.

Geometry distortion comparisons are presented in Figs. 11 and 12, for some of the PCs, over different values of s. The plots derived from the LUT method typically replicate the NNI, but with considerably less distortion.

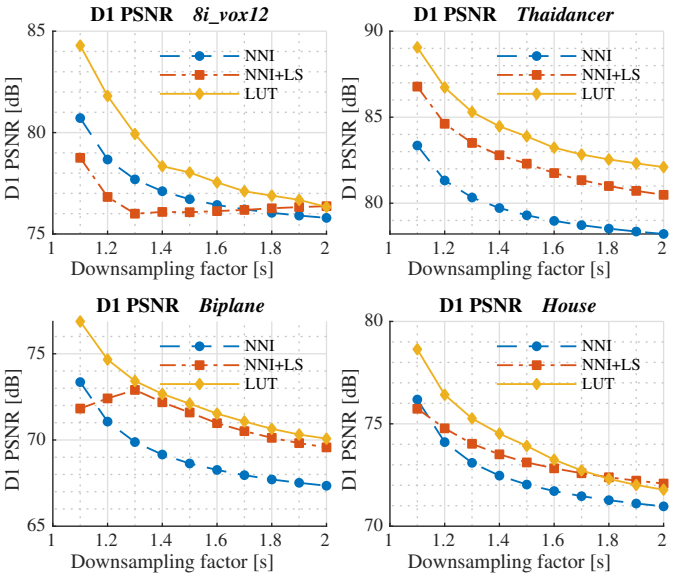


Fig. 11: $D1_{PSNR}$ metric for PCs (b), (c), (h) and (l).

Y-PSNR plots are shown in Fig. 13, where it can be seen that the proposed method outperforms the others for all PCs

TABLE II: Average gain over the NNI inside the interval $1.1 \leq s < 2$.

Point clouds		$D1_{PSNR}$ [dB]		$D2_{PSNR}$ [dB]		Y-PSNR [dB]		PPSNR [dB]		PSSIM		PVIFp	
		NNI+LS	LUT	NNI+LS	LUT	NNI+LS	LUT	NNI+LS	LUT	NNI+LS	LUT	NNI+LS	LUT
(a)	*8i_vox10*	3.27	**5.47**	4.29	**5.91**	0.17	**1.95**	2.06	**3.80**	0.02	**0.03**	**0.06**	0.04
(b)	*8i_vox12*	-0.63	**1.57**	1.23	**2.73**	-2.79	**2.84**	-1.60	**0.26**	-0.13	**-0.02**	-0.16	**0.02**
(c)	*Thaidancer*	2.85	**4.57**	3.67	**5.26**	0.14	**2.92**	2.42	**4.28**	0.01	**0.02**	**0.06**	0.05
(d)	*owlii*	3.39	**6.47**	4.41	**6.77**	0.04	**1.61**	2.37	**3.97**	0.01	**0.02**	0.05	0.05
(e)	*queen*	3.18	**6.24**	4.66	**7.12**	-0.99	**0.76**	1.27	**3.84**	0.01	**0.02**	0.05	**0.08**
(f)	*MVUB*	2.72	**3.98**	4.10	**4.80**	-0.37	**1.31**	0.12	**1.72**	0.01	**0.02**	0.02	**0.03**
(g)	*Head*	2.86	**4.32**	4.23	**5.42**	-0.69	**-0.12**	0.01	**1.43**	0.01	**0.04**	-0.03	-0.03
(h)	*Biplane*	2.10	**3.25**	3.41	**4.22**	-0.82	**-0.28**	-0.54	**0.86**	-0.01	**0.01**	-0.05	-0.05
(i)	*Statue_Klimt*	**0.88**	0.75	**1.75**	1.16	-0.94	**1.29**	-0.55	**0.30**	-0.02	**0.01**	-0.06	**-0.03**
(j)	*Arco_Valentino*	0.00	**0.93**	0.03	**1.02**	-3.59	**0.01**	-0.09	**1.02**	-0.01	**-0.03**	**-0.08**	-0.30
(k)	*Facade_00009*	0.61	**1.43**	1.88	**2.29**	-1.61	**1.58**	-0.93	**0.04**	-0.05	**-0.01**	-0.12	**-0.06**
(l)	*House*	0.89	**1.64**	2.18	**2.77**	-1.16	**0.74**	-0.79	**-0.04**	-0.02	**-0.01**	-0.09	**-0.04**

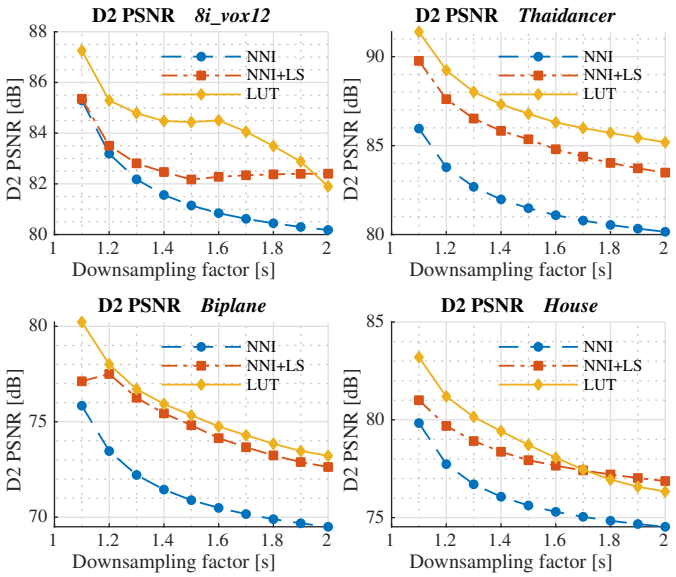


Fig. 12: $D2_{PSNR}$ metric for PCs (b), (c), (h) and (l).

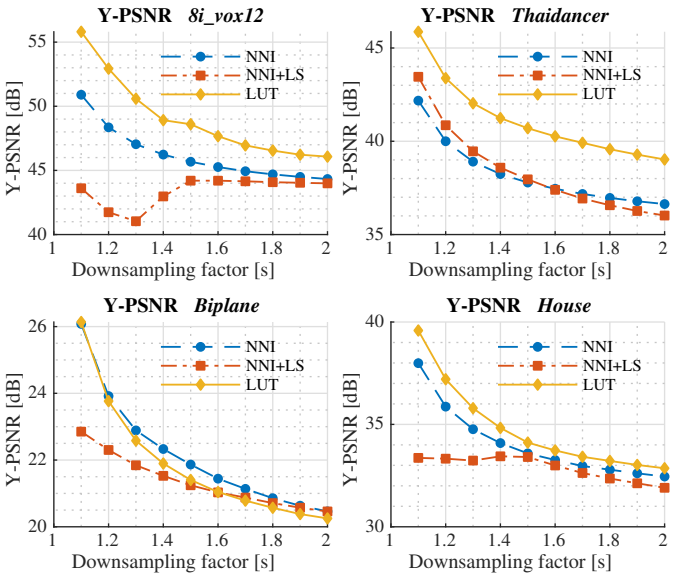


Fig. 13: Y-PSNR metric for PCs (b), (c), (h) and (l).

but *Biplane*. Note that the performance of texture metrics is more affected by content, while the performance of geometry metrics is more dependent on the PC acquisition source.

Plots with projection-based metrics are shown in Figs. 14, 15 and 16. Unlike the previous metrics, the rendering choice plays a crucial part in the projections and in the subjective evaluation. Holes caused by missing occupied children affect more these metrics than the others, which occurs more frequently for the sparser clouds.

Viewpoint projections comparing the ground truth (GT) with the three upsampling methods for *redandblack_vox10_1550* and *Biplane* are shown in Fig. 17 for visual comparison.

We ran additional tests using PCs (a), (f), (g) and (h) for $s > 2$, restricted to the $D1_{PSNR}$ metric, as shown in Fig. 18. We used the cascading method to achieve those scale factors for the LUT approach.

E. Analysis

As expected from Sec. IV-B, the "well-behaved" PCs (a), (c), (d), and (e) were the ones that presented the best performance for the proposed method.

There are odd cases such as occluded voxels with a different color than their neighbors (*queen*), cropped edges (*MUVB*), and noisy geometry scans, which degrade the performance of the NNI+LS and the LUT approaches.

The NNI+LS method is significantly affected by noisy and/or sparse geometries. In such cases, and for small scale factors, its averaging approach causes a slight shift in some voxels, which degrades distortion metrics. This behavior is illustrated for $D1_{PSNR}$ and $D2_{PSNR}$ metrics for *Biplane* and for *8i_vox12*. As s increases and the NNI becomes more uniform over all voxels, this shift in voxels is reduced, and the method's performance improves. This behavior is accentuated due to the way point metrics are calculated, by using the maximum rather than the average between the two-way measurements.

The poor performance in Y-PSNR for *Biplane* is due to the significant amount of noise in its texture, which is hard to replicate using texture interpolation solutions based on smoothing neighboring colors. It is interesting to notice the under-performance of texture for the NNI+LS method. For lower values of s, many voxels already have the correct color and applying a smoothing filter degrades the overall texture.

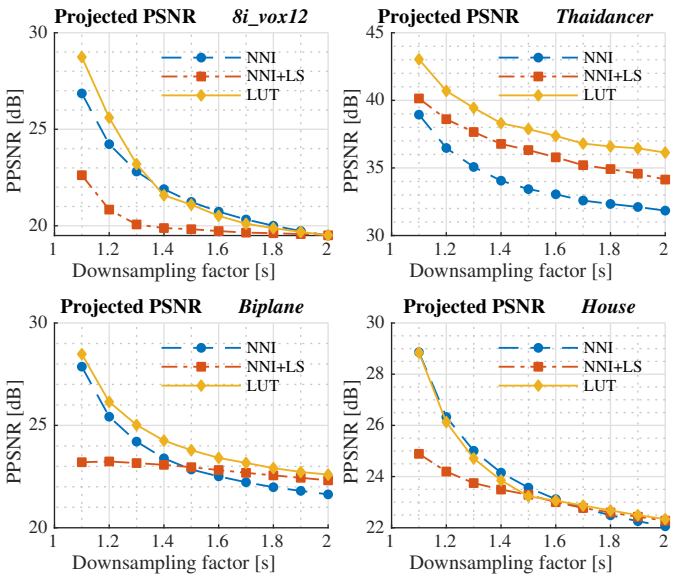


Fig. 14: Projected PSNR metric for PCs (b), (c), (h) and (l).

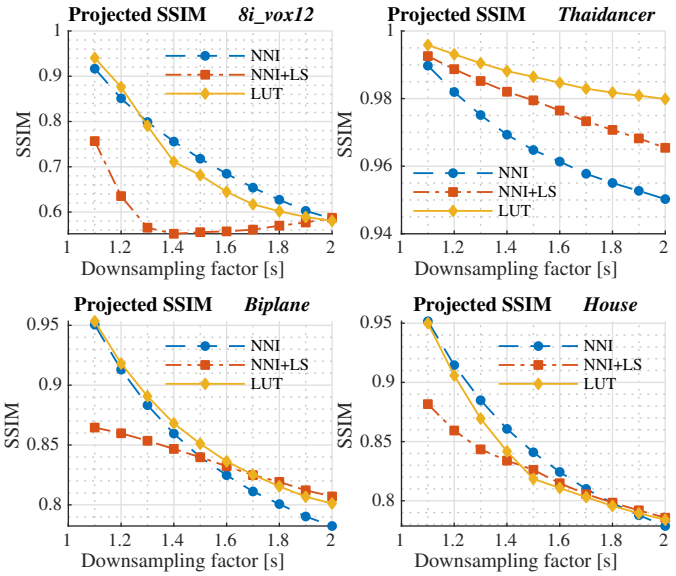


Fig. 15: Projected SSIM metric for PCs (b), (c), (h) and (l).

This is even worse for sparse PCs, where fewer neighbors are available, which may cause abrupt changes of colors. For $s > 1.5$, texture metrics for NNI and NNI+LS approaches are comparable, but since the latter was worse for smaller s, it ended up being worst overall on average. The main causes for this are noise and the absence of texture information in the neighborhoods for sparse PCs.

PPSNR results share some correlation with point-based metrics since all of those are distance-based metrics, however, the former evaluates the PC as a whole differently from the latter, which separately considers geometry and attributes. The low-values of PPSNR observed, particularly for sparse PCs, could be increased if we changed the splat size, thus making those PCs look denser, more comparable with the outputs from the upsampling methods. PSSIM and PVIFp are

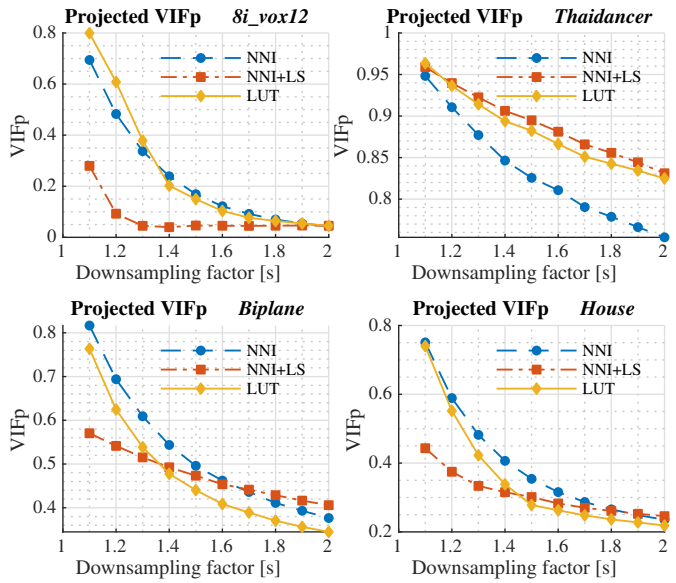


Fig. 16: Projected VIFp metric for PCs (b), (c), (h) and (l).

optimized for natural images, so it is hard to analyze their results for the chosen rendering choice, since upsampling and rendering artifacts are mixed. Thus, they cannot capture well the distortions from the outputs of the upsampling methods in sparse PC projections. They show a slight preference for the denser clouds from the NNI approach, which resemble more natural images, but are very content-depended, as pointed out in [36]. For a more meaningful understanding of how these metrics behave in the SR context, the rendering approach should be changed and subjective tests performed.

The projections of Fig. 17 show that the PCs resulting from the NNI are "bulky" and their texture "blocky", with a steep increase in the number of voxels. There are some improvements for the NNI+LS method, but the geometry is still jagged, and the number of voxels is still large. We can see a large improvement for the LUT method, with much of the aliasing removed, a finer texture, and a much more acceptable number of voxels.

Although we can still observe some gains in the geometry for the LUT method over the NNI for $s > 2$, in Fig. 18, the output PCs have too many points, and in such cases, it should be evaluated if the added complexity is worth the moderate improvements to a very degraded input geometry. The dips observed at $s = 4$ and $s = 8$ occur because of the use of consecutive $s = 2$ upsamplings, which is the least effective one for geometry carving. For other values of s, carving is more aggressive, and results are better. This is not ideal and is a point for improvement.

In Table III, we show the time profiling for some methods and PCs. The current implementation was developed using Matlab®/Octave® on a personal computer with Intel Core i7-8550U CPU @ 1.80GHz, with cache size of 8MB and 16GB of RAM. It is possible to speed up the code by removing or reducing the data augmentation step, but with poorer results. For example, over 90% of complexity reduction can be achieved with complete removal of data augmentation,

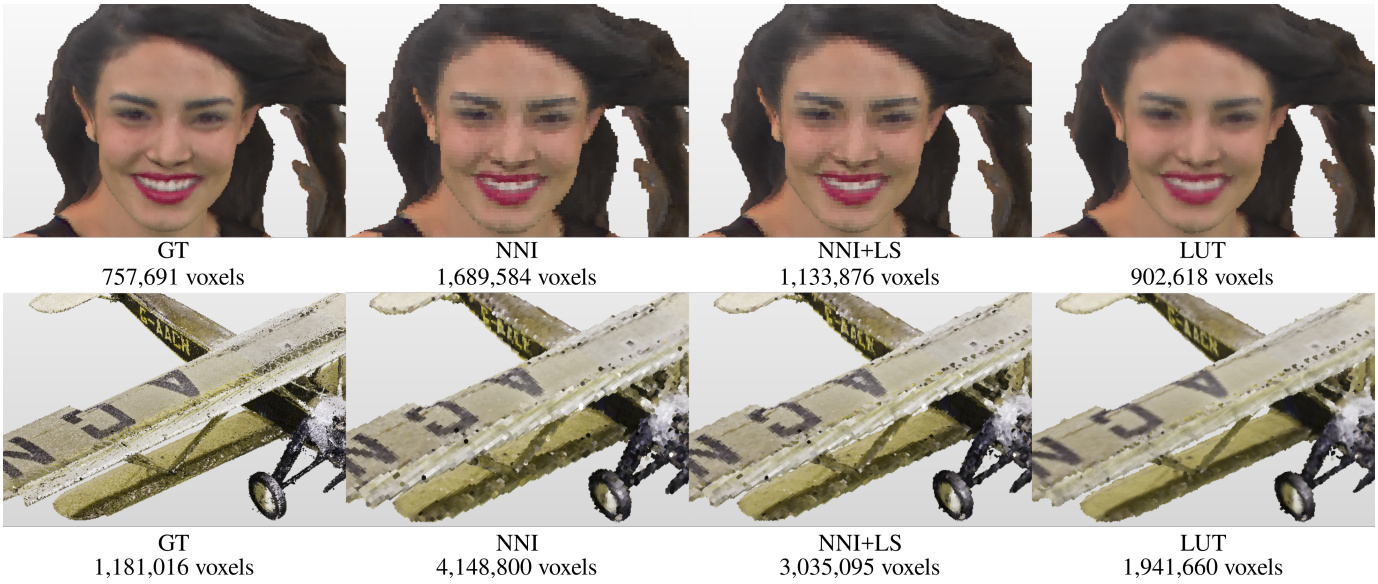


Fig. 17: PC Projections. On the top: *redandblack_vox10_1550* for $s = 2$. On the bottom: *Biplane* for $s = 4$.

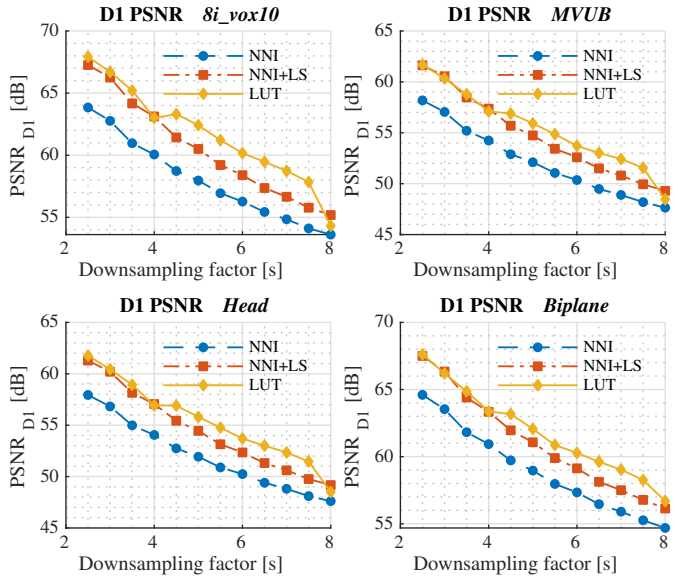


Fig. 18: D1 metric for PCs (a), (f), (g) and (h), for $s > 2$.

TABLE III: Time profiling for the tested methods.

Point Cloud	s	Time [s] NNI	LS	LUT	WAAN	Output points NNI	NNI+LS	LUT
Longdress 857,966	1.5	0.5	95.6	60.0	3.6	1,556,255	1,032,233	929,587
	2.5	0.7	75.1	128.9	7.6	2,523,083	1,564,587	1,096,984
	5	1.4	171.3	129.6	10.3	5,016,000	3,588,487	1,595,573
Andrew 279,664	1.5	0.2	11.2	17.6	1.1	456,393	302,379	303,353
	2.5	0.2	17.9	35.0	2.2	699,850	461,764	370,598
	5	0.3	83.3	35.3	3.2	1,316,050	957,647	562,290
Head 938,112	1.5	0.9	89.6	63.5	4.3	1,588,662	1,037,831	1,012,341
	2.5	0.7	78.9	131.1	8.3	2,509,250	1,647,976	1,179,319
	5	3.3	168.0	134.3	11.4	4,893,050	3,566,336	1,747,442
Biplane 1,181,016	1.5	0.6	58.0	76.8	6.0	1,898,394	1,294,955	1,316,698
	2.5	0.8	91.8	167.1	11.2	2,845,701	1,916,829	1,541,756
	5	3.4	193.1	147.3	14.0	5,004,325	3,867,418	2,193,157

with an average penalty of 2dB on $D1_{PSNR}$.

V. Conclusions

In this work, we presented a method for super-resolving voxelized PCs at fractional scales, primarily targeted for $1 < s \leq 2$, in which self-similarities at lower scales are used to define which of the possible children should be occupied. Extensive results show that the proposed method yields lower distortion results when compared to upsampling by NNI, followed or not by smoothing. This is particularly valid for point-based geometry-distortion quality metrics. For projection-based quality metrics, a broader quality assessment should be performed in order to better investigate if the subjective preference is maintained. Although only static point-clouds were used, the proposed method can be transferred to dynamic ones, probably with even better results, as more frames are available to create the LUT. Future work may focus on robustness against outlier regions in the PC. We also plan to improve the super-resolution of sparse PCs and of attributes.

References

[1] D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, and A. Tabatabai, "An overview of ongoing point cloud compression standardization activities: video-based (V-PCC) and geometry-based (G-PCC)," *APSIPA Trans. Signal and Inf. Process.*, vol. 9, 2020.

[2] S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P. Cesar, P. A. Chou, R. A. Cohen, M. Krivokuća, S. Lasserre, Z. Li, J. Llach, K. Mammou, R. Mekuria, O. Nakagami, E. Siahaan, A. Tabatabai, A. M. Tourapis, and V. Zakharchenko, "Emerging MPEG Standards for Point Cloud Compression," *IEEE J. Emerg. Sel. Top. Circuits Syst.*, vol. 9, no. 1, pp. 133–148, 2019.

[3] D. Meagher, "Geometric modeling using octree encoding," *Computer Graphics and Image Processing*, vol. 19, no. 2, pp. 129–147, Jun 1982.

[4] E. Pavez, P. A. Chou, R. L. de Queiroz, and A. Ortega, "Dynamic polygon clouds: representation and compression for VR/AR," *APSIPA Trans. Signal and Inf. Process.*, vol. 7, 2018.

[5] R. L. de Queiroz and P. A. Chou, "Compression of 3D point clouds using a region-adaptive hierarchical transform," *IEEE Trans. Image Process.*, vol. 25, no. 8, pp. 3947–3956, aug 2016.

[6] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva, "Computing and rendering point set surfaces," *IEEE Trans. Vis. Comput. Graphics*, vol. 9, no. 1, pp. 3–15, 2003.

[7] G. Guennebaud and M. Gross, "Algebraic point set surfaces," *ACM Trans. Graph.*, vol. 26, no. 3, p. 23, jul 2007.

[8] A. C. Öztireli, G. Guennebaud, and M. Gross, "Feature Preserving Point Set Surfaces based on Non-Linear Kernel Regression," *Comput. Graph. Forum*, vol. 28, no. 2, pp. 493–501, apr 2009.

[9] H. Huang, S. Wu, M. Gong, D. Cohen-Or, U. Ascher, and H. R. Zhang, "Edge-Aware Point Set Resampling," *ACM Trans. Graph.*, vol. 32, no. 1, Feb. 2013.

[10] A. Hamdi-Cherif, J. Digne, and R. Chaine, "Super-Resolution of Point Set Surfaces Using Local Similarities," *Computer Graphics Forum*, vol. 37, no. 1, pp. 60–70, jun 2017.

[11] C. Dinesh, G. Cheung, and I. V. Bajić, "3D point cloud super-resolution via graph total variation on surface normals," in *IEEE Intl. Conf. Image Process. (ICIP)*. IEEE, sep 2019.

[12] C. Dinesh, G. Cheung, and I. V. Bajić, "Super-Resolution of 3D Color Point Clouds Via Fast Graph Total Variation," in *IEEE Intl. Conf. Acoust., Speech, Signal Process. (ICASSP)*, 2020, pp. 1983–1987.

[13] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng, "PU-net: Point cloud upsampling network," in *IEEE/CVF Conf. on Comput. Vision and Pattern Recog. (CVPR)*. IEEE, jun 2018.

[14] ——, "EC-net: An edge-aware point set consolidation network," in *European Conf. on Computer Vision (ECCV)*. Springer Intl. Publishing, 2018, pp. 398–414.

[15] W. Yifan, S. Wu, H. Huang, D. Cohen-Or, and O. Sorkine-Hornung, "Patch-Based Progressive 3D Point Set Upsampling," in *IEEE/CVF Conf. on Comput. Vision and Pattern Recog. (CVPR)*. IEEE, jun 2019.

[16] R. Li, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng, "PU-GAN: a point cloud upsampling adversarial network," in *IEEE Intl. Conf. on Computer Vision (ICCV)*, Oct. 2019.

[17] H. Wu, J. Zhang, and K. Huang, "Point Cloud Super Resolution with Adversarial Residual Graph Networks," *BMVC 2020*, 2020.

[18] G. Qian, A. Abualshour, G. Li, A. Thabet, and B. Ghanem, "PU-GCN: Point cloud upsampling using graph convolutional networks," in *IEEE/CVF Conf. on Comput. Vision and Pattern Recog. (CVPR)*, 2021.

[19] Y. Qian, J. Hou, S. Kwong, and Y. He, "PUGeo-net: A geometry-centric network for 3d point cloud upsampling," in *Computer Vision – ECCV 2020*. Springer Intl. Publishing, 2020, pp. 752–769.

[20] S. Ye, D. Chen, S. Han, Z. Wan, and J. Liao, "Meta-PU: An arbitrary-scale upsampling network for point cloud," *IEEE Trans. Vis. Comput. Graphics*, pp. 1–1, 2021.

[21] D. C. Garcia, T. A. Fonseca, R. U. Ferreira, and R. L. de Queiroz, "Geometry Coding for Dynamic Voxelized Point Clouds Using Octrees and Multiple Contexts," *IEEE Trans. Image Process.*, vol. 29, pp. 313–322, 2019.

[22] D. C. Garcia, T. A. Fonseca, and R. L. de Queiroz, "Example-Based Super-Resolution for Point-Cloud Video," in *IEEE Intl. Conf. Image Process. (ICIP)*. IEEE, 2018, pp. 2959–2963.

[23] M. Corsini, P. Cignoni, and R. Scopigno, "Efficient and Flexible Sampling with Blue Noise Properties of Triangular Meshes," *IEEE Trans. Vis. Comput. Graphics*, vol. 18, no. 6, pp. 914–924, 2012.

[24] A. B. Yutaka and E. B. Y. Ohtake, "A Comparison of Mesh Smoothing Methods," in *Israel-Korea BiNational Conf. on Geometric Modeling and Comput. Graph.*, 2003, pp. 83–87.

[25] 3DG, "Common test conditions for point cloud compression," ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG), Gothenburg, SE, Approved WG 11 doc. N18883, July 2019.

[26] E. d'Eon, B. Harrison, T. Myers, and P. A. Chou, "8i Voxelized Full Bodies, version 2 – A Voxelized Point Cloud Dataset," ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG), Geneva, input document m40059/M74006, January 2017.

[27] M. Krivokuća, P. A. Chou, and P. Savill, "8i Voxelized Surface Light Field (8iVSLF) Dataset," ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG), Ljubljana, input doc. m42914, July 2018.

[28] Y. Xu, Y. Lu, and Z. Wen, "Owlii Dynamic Human Textured Mesh Sequence Dataset," ISO/IEC MPEG JTC1/SC29/WG11, Macau, China, Tech. Rep. m41658, Oct. 2017.

[29] C. Loop, Q. Cai, S. Escolano, and P. Chou, "Microsoft voxelized upper bodies - a voxelized point cloud dataset," ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG), input doc. m38673/M72012, May 2016.

[30] D. Girardeau-Montaut, M. Roux, R. Marc, and G. Thibault, "Change detection on points cloud data acquired with a ground laser scanner," in *ISPRS Workshop Laser scanning*, G. Vosselman and C. Brenner, Eds., vol. XXXVI-3/W19. Enschede, NL: Intl. Society for Photogrammetry and Remote Sensing, Sep. 2005, pp. 30–35.

[31] D. Tian, H. Ochimizu, C. Feng, R. Cohen, and A. Vetro, "Geometric distortion metrics for point cloud compression," in *2017 IEEE Intl. Conf. Image Process. (ICIP)*, Sep. 2017, pp. 3460–3464.

[32] E. M. Torlig, E. Alexiou, T. A. Fonseca, R. L. de Queiroz, and T. Ebrahimi, "A novel methodology for quality assessment of voxelized point clouds," in *App. of Digital Image Process. XLI*, A. G. Tescher, Ed., vol. 10752, Intl. Society for Optics and Photonics. SPIE, 2018.

[33] R. L. de Queiroz and P. A. Chou, "Motion-Compensated Compression of Dynamic Voxelized Point Clouds," *IEEE Trans. Image Process.*, vol. 26, no. 8, pp. 3886–3895, Aug 2017.

[34] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. S. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, apr 2004.

[35] H. R. Sheikh and A. C. Bovik, "Image information and visual quality," *IEEE Trans. Image Process.*, vol. 15, no. 2, pp. 430–444, Feb 2006.

[36] E. Alexiou, I. Viola, T. M. Borges, T. A. Fonseca, R. L. de Queiroz, and T. Ebrahimi, "A comprehensive study of the rate-distortion performance in MPEG point cloud compression," *APSIPA Trans. Signal and Inf. Process.*, vol. 8, p. e27, 2019.