Compression of Plenoptic Point Cloud Attributes Using 6-D Point Clouds and 6-D Transforms

Maja Krivokuća, Ehsan Miandji, Christine Guillemot, Philip A Chou

Compression of Plenoptic Point Cloud Attributes Using 6-D Point Clouds and 6-D Transforms

Maja Krivokuća, Ehsan Miandji, Christine Guillemot, Philip A. Chou

***Abstract*—In this paper, we introduce a novel 6-D representation of plenoptic point clouds, enabling joint, non-separable transform coding of plenoptic signals defined along both spatial and angular (viewpoint) dimensions. This 6-D representation, which is built in a global coordinate system, can be used in both multi-camera studio capture and video fly-by capture scenarios, with various viewpoint (camera) arrangements and densities. We show that both the Region-Adaptive Hierarchical Transform (RAHT) and the Graph Fourier Transform (GFT) can be extended to the proposed 6-D representation to enable the non-separable transform coding. Our method is applicable to plenoptic data with either dense or sparse sets of viewpoints, and to *complete* or *incomplete* plenoptic data, while the state-of-the-art RAHT-KLT method, which is separable in spatial and angular dimensions, is applicable only to *complete* plenoptic data. The "complete" plenoptic data refers to data that has, for each spatial point, one colour for every viewpoint (ignoring any occlusions), while "incomplete" data has colours only for the *visible* surface points at each viewpoint. We demonstrate that the proposed 6-D RAHT and 6-D GFT compression methods are able to outperform the state-of-the-art RAHT-KLT method on 3-D objects with various levels of surface specularity, and captured with different camera arrangements and different degrees of viewpoint sparsity.**

I. Introduction

A 3-D point cloud is a set of points in 3-dimensional (3-D) Euclidean space, $\mathbb{R}^3$, where each point has a *spatial position* (x, y, z), and optionally other per-point *attributes*, most typically colour. The set of all the point positions is called the point cloud's *geometry* and defines the 3-D object's shape. In a standard point cloud that contains colour attributes, each point is associated with one colour, usually represented as one (R, G, B) triplet. This representation can be sufficient for a 3-D object whose surface is *Lambertian* [1] or nearly Lambertian - that is, the colour of a point on the object's surface appears the same, or almost the same, regardless of the viewpoint of the observer. But for objects whose surfaces are reflective and therefore contain areas where the colour appears different depending on the viewing angle, one colour per point is insufficient. For this reason, a new generation of point cloud has recently emerged: the *plenoptic point cloud* [2, 3, 4, 5, 6]. In a plenoptic point cloud, each spatial point (x, y, z) is associated with multiple colour values, to represent the colour of that point as viewed from different directions. This is essentially a discretised representation of the 5-D *plenoptic function* [7]:

$$P(x, y, z, \theta, \phi), \tag{1}$$

where P is the radiance observed for every possible spatial position (x, y, z), with every viewing angle (θ, ϕ), where θ is the azimuth and ϕ the elevation.

Along with the added realism and richness of visual data that is possible with a plenoptic point cloud comes the additional burden of efficiently representing and coding the multiple colour attributes associated with each point. Standard point cloud geometry and attribute compression have been studied for a number of years now (e.g., see [8] for a recent survey), and have also recently been adopted into the standardisation activities of both MPEG [9, 10] and JPEG [11]. However, the compression of *plenoptic* point clouds remains a challenging and open research problem, which has only recently begun to receive attention. The few existing plenoptic point cloud compression methods usually rely either on the fact that a dense set of captured viewpoints will be available per spatial point, normally in a uniform arrangement, or that the set of viewpoints will be *complete* - that is, each spatial point will have a colour for each viewpoint, even if that spatial point is not visible from every viewpoint. However, in real-life point cloud captures, the more likely scenario is a sparse, sometimes irregularly spaced, set of camera viewpoints being used to capture a 3-D object (or scene), with occlusions making each spatial point visible only from certain viewpoints. The existing compression algorithms also either tackle the correlation across viewpoints and across spatial points separately, e.g., RAHT-KLT [4] or the methods used for Surface Light Field compression in [5, 6], or they only tackle one of these dimensions and not the other, e.g., [12]. Other methods, such as the one in [13], project the input point cloud onto 2-D video frames and use standard video coding techniques, which may not be suitable for applications where coding the point cloud directly in its 3-D domain is more appropriate. Section II will provide a more detailed review of existing techniques for plenoptic point cloud compression. In all cases, the existing compression methods treat the spatial and angular (viewpoint) dimensions *separably*.

In this paper, we propose a novel plenoptic point cloud representation, as a 6-D point cloud, enabling efficient *joint* spatio-angular compression of its colour attributes. The 6-D representation results from embedding the last two of the 5-D plenoptic coordinates (x, y, z, θ, ϕ) into a 3-D space as the coordinates (x^a, y^a, z^a) on the unit sphere. This results in the 6-D representation (x, y, z, x^a, y^a, z^a), where the a superscript stands for "angular". We propose compression schemes for the plenoptic colour data associated with the 6-D spatio-angular points in our representation, making use of both the *Region-Adaptive Hierarchical Transform* (RAHT) [14, 15] and the *Graph Fourier Transform* (GFT) [16, 17].

We demonstrate the advantages of the proposed representation for plenoptic point cloud data with various degrees of

surface specularity, viewpoint sparsity, and camera arrangements. We also show that it allows us to handle so-called *incomplete* viewpoint data, i.e., data for which not all spatial points are visible from all viewpoints. In summary, our main contributions are as follows:

1) We introduce the notion of identifying and encoding only the *"valid"* colour data for plenoptic point clouds - we term such data *incomplete*. This means that for each spatial point, only the colours for the viewpoints from which this spatial point is *visible* are encoded. Therefore, different spatial points may have plenoptic colour vectors of different lengths, and these colour vectors may correspond to different viewpoints. This is in contrast to previous work (notably [3, 4]), where the coding of plenoptic point clouds requires *complete* viewpoint data, i.e., a colour value for every viewpoint, per spatial point, ignoring any occlusions.[1]
2) We introduce a 6-D global scene coordinate system for plenoptic point cloud representation, which enables the encoding of *incomplete* viewpoint data, by treating the plenoptic point cloud as a 6-D point cloud.
3) We show that the state-of-the-art method for *non-plenoptic* point cloud attribute coding, RAHT [14, 15], can be applied *non-separably* to our 6-D spatio-angular space to encode *plenoptic* colour values, without requiring a *complete* set of viewpoints. Therefore we contribute an extended 6-D version of the RAHT method, which is shown to achieve excellent rate-distortion performance for plenoptic datasets with various levels of surface specularity, captured from various camera arrangements and viewpoint densities.
4) We also propose an extension of the GFT compression algorithm to our 6-D spatio-angular space, and show that it is likewise able to work *non-separably* on *incomplete* plenoptic colour data to achieve similar rate-distortion performance as our 6-D RAHT. While the idea of using graphs to compress point cloud attributes is not new (e.g., [18, 19]), to the best of our knowledge the proposed method is the first time the GFT has been applied in 6-D spatio-angular space and for the purpose of compressing *plenoptic* point cloud attributes.
5) We demonstrate that the proposed 6-D representation allowing joint spatio-angular processing and compression, yields better rate-distortion performance than separate spatial and angular processing, in particular in comparison with the state-of-the-art RAHT-KLT method [4], and in the presence of high surface specularity. Similarly, we show that there is value (in terms of large rate-distortion gains) in encoding just the *incomplete* plenoptic data instead of the *complete* data.

[1]Note that throughout this paper, an *occluded* or *invisible* spatial point, relative to a given camera viewpoint, will refer to a spatial point on a 3-D object surface, which cannot be seen from that camera, because that part of the 3-D object surface is occluded from the camera's view. Hence this spatial point does not have a colour attribute corresponding to that camera direction. Also note that, for simplicity, "viewpoint" will be used somewhat interchangeably with "view direction", to indicate both, the angular direction of a capturing camera in 3-D space and also the rendering viewpoint of the 3-D object. This latter point will be explained more precisely near the end of Section VI-B.

The rest of the paper is organised as follows. In Section II, we summarise the relevant existing methods in the literature, pertaining to plenoptic point cloud or *Surface Light Field* (SLF) compression. In Section III, we introduce our proposed 6-D point cloud framework. In Sections IV and V, we explain, respectively, our extensions of the RAHT and GFT compression methods to the proposed 6-D space. In Section VI, we describe the plenoptic data that we use for testing our ideas in this paper. In Section VII, we summarise our experimental procedure and key results, in Section VIII we discuss the complexity of the proposed approach, and in Section IX we conclude the paper.

II. Related Work

In [4], the plenoptic point cloud is represented as a collection of vectors. For each surface point $\boldsymbol{p}_i$ in a finite set of points $\{\boldsymbol{p}_i | i \in [1, N_p]\}$ in $\mathbb{R}^3$ (where N_p is the total number of spatial points), with a finite number of viewpoints N_c (equal to the number of camera viewpoints that were used to capture the 3-D object), this point's spatial position and colours are represented as a vector

$$\boldsymbol{p}_i = [x_i, y_i, z_i, R_i^1, G_i^1, B_i^1, \ldots, R_i^{N_c}, G_i^{N_c}, B_i^{N_c}], \quad (2)$$

where $[R_i^1, \ldots, B_i^{N_c}]$ is the *plenoptic* or *multi-view* colour vector. The authors propose to compress these plenoptic colour vectors by four possible extensions to the *Region-Adaptive Hierarchical Transform* (RAHT) [14, 15] colour coding method: RAHT-1, RAHT-2, RAHT-KLT, and RAHT-DCT. In RAHT-1 and RAHT-2, the camera positions are projected onto a 2-D plane, then subdivided using a quadtree to obtain one camera position per square. RAHT is applied on the colours associated with each camera position in this quadtree arrangement, which produces N_c RAHT coefficients per voxel. Then RAHT is additionally applied across the voxel spatial coordinates by either considering each DC coefficient produced in the first RAHT transform as the colour of the corresponding voxel, then applying RAHT over all the voxels (RAHT-1), or by considering each of the N_c RAHT coefficients per voxel as that voxel's colour in one of N_c separate point clouds (sharing the same geometry), which are then encoded separately using RAHT (RAHT-2). In RAHT-KLT, an $N_c \times N_c$ covariance matrix is computed for each colour channel C (Y, U, V channels were used in [4]), then the eigenvectors of each covariance matrix are computed through a *Singular Value Decomposition* (SVD) and are used to perform a *Karhunen-Loève Transform* (KLT) on each colour vector $\mathbf{c}(n) = [C_1(n), C_2(n), ..., C_{N_c}(n)]^T$ for each point n in the point cloud, for each colour channel C. The $N_p \times 3$ matrix of KLT-transformed vectors for each of the N_c viewpoints is then encoded with RAHT. In RAHT-DCT, the KLT is replaced by the DCT, after first rearranging the plenoptic colours for each voxel to follow a pre-determined spiral path around a sphere. The RAHT-KLT approach was shown in [4] to produce the best results. Note, however, that all of these approaches process the spatial and angular dimensions separably. This leads to the requirement that there should be a *complete* set of viewpoints (i.e., a colour value for each

viewpoint, per spatial point, regardless of whether or not that spatial point is visible from every viewpoint). Completing the viewpoints to use these methods means that redundant data needs to be encoded, because certain spatial points will not be visible from certain viewpoints due to occlusions.

In [20], it was shown that the rate-distortion performance of RAHT-KLT can be further improved by performing a prior subdivision of the plenoptic point cloud into clusters based on similar colour values (e.g., by using *k-means*), followed by a separation of each cluster into specular and diffuse components (e.g., by using *Robust Principal Component Analysis* [21]), then coding each component separately with RAHT-KLT.

In [5, 6], the plenoptic point cloud is considered in the framework of a *Surface Light Field* (SLF) representation. The SLF can be considered a function $f(\boldsymbol{\omega}|\boldsymbol{p})$, such that for a point $\boldsymbol{p}$ on the surface, $f(\boldsymbol{\omega}|\boldsymbol{p})$ represents the colour of a light ray starting at $\boldsymbol{p}$ and emanating outwards in direction $\boldsymbol{\omega}$ [6]. Each surface point $\boldsymbol{p}$ therefore has a "view map", or "colour map", which describes the colour of $\boldsymbol{p}$ as seen from different viewpoints, similar to a *lenslet* representation in light field imaging [22]. Since in a point cloud, the (x, y, z) points are defined directly on the surface of a 3-D object, we can say that the plenoptic point cloud is equivalent to a SLF. In [5, 6], input light field images are mapped to a point cloud geometry to obtain a SLF, then the points' view maps are encoded using a B-Spline wavelet basis and the transform coefficients are compressed spatially using existing point cloud codecs. Note again that separable processing is used.

Other methods also exist in the literature, which compress Surface Light Fields, but not explicitly considering a point cloud representation. For example, in [23], the SLF is represented as a collection of local *Hemispherical Radiance Distribution Functions* (HRDFs) at each surface point of a 3-D object or scene. Spatial correlation across surface points is taken into account by clustering points with similar HRDFs using *k-means*, then per-cluster dictionaries of separable transforms are learned. The success of the separable method in [23], as well as the separable methods in [5, 6], relies on having a *complete* set of recorded view directions arranged in a predictable (uniform) manner, per spatial point. However, this may not be applicable to real-life capture scenarios, where only a sparse set of captured viewpoints may be available and those viewpoints may be irregularly spaced.

Some older algorithms exist for SLF compression as well, which propose various techniques for compressing the SLF "colour maps" (mainly using quite standard image compression techniques), but without any spatial compression, e.g., [24, 25, 26].

There also exist a few methods in the literature that apply a video-based framework to compress plenoptic point clouds. In [12], the MPEG V-PCC [9] Test Model is extended to be able to handle multiple colour values per point, but the correlations across the different viewpoints are not taken into account. In [27], the authors reduce the memory requirements of their method in [12], by encoding only a subset of all the available viewpoints for different regions of the point cloud, depending on how much surface specularity is present in each region. In [13], the authors propose to extend the MPEG V-PCC codec to plenoptic point clouds by projecting the point cloud at each viewpoint onto 2-D video frames, then using the Multiview extension of HEVC (MV-HEVC) [28] to encode the multiview attributes. However, these video-based methods may not be suitable for applications using rendering from the 3-D point cloud, where it makes more sense to directly encode the 3-D point cloud. It should also be noted that these methods, like all those mentioned earlier, process spatial and angular dimensions separably, and hence require *complete* data.

III. Proposed Framework

Let us consider a 5-D plenoptic point cloud (or Surface Light Field) consisting of spatio-angular points $\boldsymbol{p}_i = (x_i, y_i, z_i, \theta_i, \phi_i)$ and their attributes f_i, for $i = 1, \ldots, N$. Typical attributes are the colour components of each point, e.g., $f_i = (R_i, G_i, B_i)$. The spatio-angular points in the 5-D point cloud can be represented as a list as:

$$\begin{gathered} x_1, y_1, z_1, \theta_1, \phi_1 \\ \vdots \\ x_i, y_i, z_i, \theta_i, \phi_i \\ \vdots \\ x_N, y_N, z_N, \theta_N, \phi_N. \end{gathered} \tag{3}$$

As an example, illustrated in Fig. 1 (right), the first 7 spatio-angular points belong to the same surface point s_1, and the next 8 spatio-angular points belong to the surface point s_2. That is, the triples (x_i, y_i, z_i) are identical for $i = 1, \ldots, 7$, and take another value for $i = 8, \ldots, 15$. In general, the list of spatio-angular points may be partitioned into sets of angular directions, each set corresponding to a different spatial point. Each spatial point can have different angular directions, and different numbers of angular directions. Each spatio-angular point will also have colour attributes associated with it, such that it becomes an 8-D vector: $\boldsymbol{p}_i = (x_i, y_i, z_i, \theta_i, \phi_i, C1_i, C2_i, C3_i)$, where $C1$, $C2$, and $C3$ represent different colour channels in a given colour space. In this paper, we will refer to such a vector as an *8-D attributed point*.

Our objective is to compress 5-D plenoptic point clouds. There are two parts to this problem: compressing the set of spatio-angular points $(x_i, y_i, z_i, \theta_i, \phi_i)$, $i = 1, \ldots, N$ (i.e., the point cloud's *geometry* and angular directions), and compressing the per-point, per-direction *attributes* f_i, $i = 1, \ldots, N$. In our work, we assume that the geometry has already been compressed and is available to both the encoder and decoder, and that the angular directions are likewise known at the decoder.[2] So our goal is to compress the colour attributes,

[2]We make no particular assumptions about which method for geometry coding or coding of angular directions is used. In theory, any existing point cloud geometry coding method could be used for the spatial positions, e.g., see [8] and related references therein. For encoding and transmitting the angular directions, an investigation on how to best do this is outside the scope of our current manuscript, but it would make an interesting (and important) future research topic. We also do not make any assumptions on whether the coding of the spatio-angular positions should done on the 5-D points (x, y, z, θ, ϕ) (or the 6-D points (x, y, z, x^a, y^a, z^a)), or separately for the geometry and angular directions. Again, such investigations are beyond the scope of our current manuscript, but should be looked into in the future.

given the geometry and angular directions. A convenient way to represent and organise the 5-D spatio-angular points $\boldsymbol{p}_i = (x_i, y_i, z_i, \theta_i, \phi_i)$, having any real value for the spatial coordinates (x_i, y_i, z_i) and the angular coordinates (θ_i, ϕ_i), is to embed the 2-D angular coordinates into 3-D coordinates on the surface of a sphere, and then to hierarchically partition the resulting 6-D space using Morton codes.

A. 6-D Representation of the 5-D Point Cloud

We embed the 2-D space of angular coordinates (θ, ϕ) into a 3-D space of coordinates (x^a, y^a, z^a) on the surface of a sphere, in order to preserve the angular geometry independent of the coordinate system. Specifically, for each view direction (θ, ϕ), we compute the coordinates (x^a, y^a, z^a) on the surface of a unit sphere as:

$$\begin{aligned} x^a &= \cos(\phi)\cos(\theta), \\ y^a &= \cos(\phi)\sin(\theta), \\ z^a &= \sin(\phi). \end{aligned} \tag{4}$$

Note that the "a" in (x^a, y^a, z^a) above stands for "angular", to differentiate these angular coordinates from the spatial (x, y, z) coordinates of the surface points.

B. Morton Code Computation

Morton codes provide an ordering along a Z-shaped space-filling curve [29], allowing multi-dimensional data to be mapped to one dimension while preserving locality of the data points. This makes them convenient to use for hierarchical subdivision of space, e.g., into quadtrees or octrees, or higher-dimensional blocks (e.g., see Section V-A). Here, we apply them to our 6-D space of spatio-angular points. In order to compute a Morton code for each spatio-angular point, we need to ensure first that the spatial and angular coordinates fit into a suitable integer range. This generally requires a rescaling, translation, and rounding of the original spatial and angular coordinates. For each Cartesian coordinate c_i (where c_i represents either x_i, y_i, z_i, x_i^a, y_i^a, or z_i^a), and given a rescaling range of $[c_{min}, c_{max}]$, c_i can be rescaled, translated, and rounded into the range $[c_{min}, c_{max}]$ as:

$$c_i{}' = \left\lfloor \frac{(c_{max} - c_{min}) \times (c_i - \min \boldsymbol{c})}{Range_{max}} + c_{min} \right\rfloor, \tag{5}$$

where $\min \boldsymbol{c}$ indicates the minimum of all the corresponding c values (e.g., if c represents the spatial coordinate x, then this would be the minimum of all the input x values), and $Range_{max}$ represents the maximum out of the input x, y and z ranges (if c is a spatial coordinate) or the maximum out of the input x^a, y^a and z^a ranges (if c is an angular coordinate). For example, if c is a spatial coordinate, then:

$$\begin{aligned} &Range_{max} \\ &= \max\left((\max \boldsymbol{x} - \min \boldsymbol{x}), (\max \boldsymbol{y} - \min \boldsymbol{y}), (\max \boldsymbol{z} - \min \boldsymbol{z})\right). \end{aligned} \tag{6}$$

We do this to ensure that the scaling is uniform across all three axes in the spatial dimension and all three axes in the angular dimension, thereby preserving the aspect ratio of the input 3-D object. Note that the rescaling range $[c_{min}, c_{max}]$ may be different for the spatial and angular coordinates.

A separate Morton code M_i is computed for the integer version of each input 6-D spatio-angular point (computed as explained above), by interleaving the bits of the corresponding 6-D vector $\boldsymbol{p}_i{}' = (x_i{}', y_i{}', z_i{}', x_i^{a\prime}, y_i^{a\prime}, z_i^{a\prime})$ as follows:

1) We need to use the same number of bits for each of the 6 coordinates in $\boldsymbol{p}_i{}'$, to enable correct interleaving of the bits for the Morton code. So, the total Morton code length will be equal to 6 times the maximum number of bits used for any coordinate in $\boldsymbol{p}_i{}'$. For example, if the integer spatial coordinates $(x_i{}', y_i{}', z_i{}')$ are in the range [0, 255] and the integer angular coordinates $(x_i^{a\prime}, y_i^{a\prime}, z_i^{a\prime})$ are in the range [0, 7], then the maximum number of bits needed for any coordinate is 8, so the length of each Morton code should be $6 \times 8 = 48$ bits. To be represented in a long integer on a modern computer, the maximum possible length of a Morton code is 64 bits, limiting the number of bits of each of the six coordinates in $\boldsymbol{p}_i{}'$ to 10 bits.
2) For each $\boldsymbol{p}_i{}'$, compute its Morton code by starting with the MSB (most significant bit) of $x_i{}'$, followed by the MSB of $y_i{}'$, then the MSBs of $z_i{}'$, $x_i^{a\prime}$, $y_i^{a\prime}$, and $z_i^{a\prime}$, in that order. Then move on to the second MSB of $x_i{}'$, followed by the second MSB of $y_i{}'$, and so on, until we have accounted for all the bits of each of the 6 coordinates.
3) Repeat steps 1 and 2 for each input 6-D vector $\boldsymbol{p}_i{}'$ (where $i = 1, ..., N$). In the end, there will be N Morton codes in total, and each Morton code will be an integer.
4) Sort the computed Morton codes in ascending order, and sort the corresponding 6-D $\boldsymbol{p}_i{}'$ vectors in the same order.
5) Some Morton codes may be duplicated. This is because some 6-D vectors $\boldsymbol{p}_i{}'$ may end up being identical after rescaling of their original coordinates into a fixed range. Since we want to have only a unique set of Morton codes (i.e., a unique set of spatio-angular points), we average the coordinate and attribute values for the spatio-angular points which have the same Morton code. The final set of 6-D spatio-angular points may therefore be smaller than the original number of spatio-angular points in the input data. In this case, there would inherently be some loss after compression, as we would not be able to recover exactly the colour values of all the original (before rescaling) spatio-angular points, whose coordinates and colours were averaged before compression (see Section VII for further explanation).

Note that the Morton codes are computed firstly for a *complete* set of viewpoints, i.e., given a set of all the captured viewpoints, and all the spatial points, we compute the Morton code for every possible spatio-angular point. Afterwards, we discard the Morton codes that correspond to the "invalid" spatio-angular points, i.e., those for which the corresponding spatial point is not visible from the corresponding viewpoint. We term the resulting set of 6-D points, plus their corresponding attribute values, *incomplete* viewpoint data.

IV. 6-D Spatio-Angular RAHT Extension

The *Region-Adaptive Hierarchical Transform* (or RAHT) [14, 15] and its variants [30, 31, 32, 33, 34, 19] are the

current state-of-the-art method for point cloud attribute coding, recently having been integrated into the MPEG Point Cloud Coding (G-PCC) standard [9, 10].

Although there are many ways to understand RAHT (e.g., [30, 34]), perhaps most relevant to this paper is to understand RAHT as a transform on a binary tree-structured subdivision of space into rectangular cuboids, or blocks $B_{b,\boldsymbol{m}}$ comprising all points in space whose Morton codes have a b-bit prefix equal to $\boldsymbol{m} = m_0 m_1 \cdots m_{b-1}$. Block $B_{b,\boldsymbol{m}}$ is the *parent* of its *child* blocks $B_{b+1,\boldsymbol{m}0}$ and $B_{b+1,\boldsymbol{m}1}$, which are *siblings*. Thus blocks are identified with nodes in a binary tree. Block $B_{b,\boldsymbol{m}}$ corresponds to a node at level b of the tree, whose path from the root node is given by the block's b-bit Morton prefix $\boldsymbol{m}$. The root of the tree is at level $b = 0$, while the leaves of the tree are at level $b = b_{\max}$. The leaves of the tree correspond to the points in the point cloud, ordered by their Morton codes. A block at any level that contains no points in the point cloud are pruned from the tree, as they are *unoccupied*. All remaining blocks are *occupied*. The *weight* $w_{b,\boldsymbol{m}}$ of a block $B_{b,\boldsymbol{m}}$ is the number of points in the point cloud that are contained by the block, i.e., the number of points in the point cloud that have the b-bit prefix $\boldsymbol{m}$.

RAHT's forward transform proceeds from the leaves to the root, while its inverse transform proceeds from the root to the leaves. In the forward transform, for each block $B_{b,\boldsymbol{m}}$ with occupied children $B_{b+1,\boldsymbol{m}0}$ and $B_{b+1,\boldsymbol{m}1}$, RAHT computes the Givens rotation

$$\begin{bmatrix} F_{b,\boldsymbol{m}} \\ G_{b,\boldsymbol{m}} \end{bmatrix} = \begin{bmatrix} a & b \\ -b & a \end{bmatrix} \begin{bmatrix} F_{b+1,\boldsymbol{m}0} \\ F_{b+1,\boldsymbol{m}1} \end{bmatrix}, \tag{7}$$

where

$$a = \sqrt{\frac{w_{b+1,\boldsymbol{m}0}}{w_{b+1,\boldsymbol{m}0} + w_{b+1,\boldsymbol{m}1}}} \quad b = \sqrt{\frac{w_{b+1,\boldsymbol{m}1}}{w_{b+1,\boldsymbol{m}0} + w_{b+1,\boldsymbol{m}1}}}. \tag{8}$$

Note that $a^2 + b^2 = 1$ and hence (7) is an orthonormal transform. The coefficients $F_{b_{\max},\boldsymbol{m}}$ at the leaves are initialized to the attributes of the points. It can be seen by induction that $F_{b,\boldsymbol{m}}$ for any block $B_{b,\boldsymbol{m}}$ is equal to the average of the attributes of the points in $B_{b,\boldsymbol{m}}$, scaled up by the square root of the number of points in $B_{b,\boldsymbol{m}}$. Thus $F_{0,\emptyset}$ is the DC coefficient of the transform, while $\{G_{b,\boldsymbol{m}}\}$ are the AC coefficients. The entire transform is orthonormal because it is composed of orthonormal Givens rotations. Hence all coefficients may be quantized in the transform domain without blowing up the quantization error in the signal domain. The inverse transform simply computes coefficients from the root to the leaves using the inverse of (7).

To extend RAHT to our 6-D spatio-angular space, it is sufficient to simply extend the Morton codes to 6 dimensions. Instead of the Morton codes representing only the (integer) spatial coordinates of the input point cloud, they now represent the (integer) *spatio-angular* coordinates, as detailed in Section III-B. The inputs to RAHT are then the sorted 6-D Morton codes (with duplicates and "invalid" points removed) and the colours of the corresponding 6-D spatio-angular points. The resulting RAHT transform coefficients are uniformly quantized, sorted according to their weights as in [15], and encoded (separately for each colour channel) by using the *Run-Length Golomb-Rice* (RLGR) entropy coder [35].

V. 6-D Spatio-Angular Graph Fourier Transform

In this section, we explain the extension of the well-known *Graph Fourier Transform* (GFT) to our proposed 6-D spatio-angular space, and the application of this 6-D GFT to the compression of colour vectors in plenoptic point clouds.

A. 6-D Block Subdivision

We use the Morton codes computed in Section III-B to subdivide the input plenoptic point cloud into blocks in 6-D space defined by the integer spatial coordinates (x', y', z') and the integer angular coordinates $(x^{a\prime}, y^{a\prime}, z^{a\prime})$. The 6-D spatio-angular points whose Morton codes have the same prefix (i.e., the same MSBs) belong in the same 6-D block. The more significant bits that we consider as the prefix, the smaller the 6-D blocks will be. If we wish to split along each of the six axes in 6-D space, the number of most significant bits that we consider must be a multiple of 6. The block subdivision is necessary in order to provide reasonably-sized subspaces on which to construct the graphs that we will use for efficient representation and compression of the plenoptic colour data. Note that we process only the *occupied* 6-D blocks.

B. Blockwise 6-D Graph Construction

We construct a separate graph in each of the 6-D blocks described in Section V-A. For a block containing a set of nodes $V = \{v_i, i = 1, ..., n\}$ and a set of edges $E = \{e_j, j = 1, ..., m\}$, we construct a graph $G = (V, E)$. Each node v_i of the graph represents the 6-D spatio-angular location $(x_i', y_i', z_i', x_i^{a\prime}, y_i^{a\prime}, z_i^{a\prime})$ of the corresponding 6-D point that is found inside the same block. Note that since we wish to use the graph for point cloud *attribute* compression, we consider only the spatio-angular coordinates $(x_i', y_i', z_i', x_i^{a\prime}, y_i^{a\prime}, z_i^{a\prime})$, and not the colour values, for the construction of the graph. The colours are considered signals defined over the graph nodes. We assume that the geometry data has already been compressed and transmitted separately to the decoder, and that the decoder knows the locations of the viewpoints, so that the graphs can be constructed independently at the encoder and decoder ends. Also note that the spatio-angular points corresponding to the graph nodes in each block are only the "valid" points, representing *incomplete* viewpoint data (see Section III-B).

Within each 6-D block, we create a *fully connected* graph. This means that every graph node (spatio-angular point) inside the block is connected to every other graph node inside the same block. The graph edges are also *weighted*, and the weight of the edge between two nodes is computed as the inverse of the total squared distance d^2 between them (i.e., $1/d^2$). To compute the squared distance $d(v_i, v_k)^2$ between two graph nodes v_i and v_k (where $i \neq k$), we consider a linear combination of the squared Euclidean distance between these nodes'

spatial coordinates, $d_s(v_i, v_k)^2$, and the squared Euclidean distance between their angular coordinates, $d_a(v_i, v_k)^2$, as:

$$\begin{gathered} d_s(v_i, v_k)^2 = (x_i{}' - x_k{}')^2 + (y_i{}' - y_k{}')^2 + (z_i{}' - z_k{}')^2, \\ d_a(v_i, v_k)^2 = (x_i^{a\prime} - x_k^{a\prime})^2 + (y_i^{a\prime} - y_k^{a\prime})^2 + (z_i^{a\prime} - z_k^{a\prime})^2, \\ d(v_i, v_k)^2 = d_s(v_i, v_k)^2 + d_a(v_i, v_k)^2. \end{gathered} \tag{9}$$

The $n \times n$ *adjacency matrix* $\mathbf{A}$ in each 6-D block therefore consists of the inverse distance values $1/d(v_i, v_k)^2$ in locations (i, k) and (k, i) where the corresponding nodes v_i and v_k are connected to each other. If a block contains only one node, its adjacency matrix is set to 0. The $n \times n$ *degrees matrix* $\mathbf{D}$ in each 6-D block is computed as the sum of the weights in $\mathbf{A}$ across each row (or column). These sums are placed on the main diagonal of $\mathbf{D}$, while other locations in $\mathbf{D}$ have values of 0. Finally, the $n \times n$ *graph Laplacian matrix* $\mathbf{L}$ in each 6-D block is computed as:

$$\mathbf{L} = \mathbf{D} - \mathbf{A}. \tag{10}$$

Note that $\mathbf{A}$, $\mathbf{D}$, and therefore $\mathbf{L}$, are all *symmetric* matrices. This means that every row and column of $\mathbf{L}$ sums to zero.

C. Graph Laplacian Analysis and Synthesis

Both the analysis and synthesis of the input plenoptic attribute data are performed by using an *orthonormal basis* consisting of the unit-norm eigenvectors of the graph Laplacian matrix computed in (10). Note that since the encoder and decoder are both assumed to have access to the input point cloud geometry and viewpoint locations a priori, the Laplacian eigenvectors can be computed independently at each end. The graph Laplacian matrix $\mathbf{L}$ is *positive semi-definite*, meaning that all of its eigenvalues are non-negative and real. The number of eigenvalues with a value of 0 indicates the number of *connected components* in the corresponding graph. Since we use a fully connected graph inside each 6-D block, the graph for each block has only one connected component (and therefore one DC coefficient – see below).

We sort the set of Laplacian eigenvalues for each 6-D block in order of increasing magnitude. We then sort their corresponding eigenvectors in the same order. Since the Laplacian eigenvalues are considered analogous to the frequencies of their corresponding basis vectors (eigenvectors) [36, 37], this arrangement effectively puts all the lowest-frequency basis vectors first, followed by increasingly higher-frequency basis vectors. The eigenvectors corresponding to the 0 eigenvalue can thus be considered to represent "DC" basis vectors.

At the encoder, we decompose the $n \times 3$ matrix of colour vectors, $\mathbf{c}$, corresponding to the n graph nodes within a 6-D block, onto the basis of $n \times n$ unit-norm eigenvectors $\mathbf{\Phi}$ for that block, ordered according to their eigenvalues as explained above. This produces a set of $3 \times n$ *spectral coefficients* (n coefficients per colour channel), γ, for this block:

$$\gamma = \mathbf{c}^T \mathbf{\Phi}. \tag{11}$$

For the work in this paper, prior to entropy coding, we first uniformly quantize all the spectral coefficients resulting from the 6-D GFT, then we order the quantized coefficients according to the 6-D blocks' Morton codes. The DC coefficients (for all the blocks, in Morton order) are placed first, followed by the AC coefficients (for all the blocks, in Morton order). We then entropy-code the ordered coefficients separately per colour channel, using the RLGR entropy coder [35] similary as for 6-D RAHT (see Section IV).

At the decoder, the recovered set of all the dequantized GFT coefficients $\hat{\gamma}$ are used to reconstruct the colour values for the graph nodes in the corresponding 6-D block:

$$\hat{\mathbf{c}}^T = \hat{\gamma} \mathbf{\Phi}^T. \tag{12}$$

VI. Plenoptic Data Preparation

In order to test our ideas, we consider synthetic Surface Light Field data with an HRDF representation [23], which simulates complex surface reflectance properties on different levels of surface roughness, with different-coloured light sources that reflect off the object surfaces. Such data offers much more complexity to test our proposed ideas than, for example, the 8iVSLF data [2], which does not have a lot of variation in colour across the different viewpoints, has light sources with only a single colour (white), and does not enable us to vary the surface specularity or the viewpoint density. By placing the HRDF data into our proposed 6-D representation, we are able to simulate various realistic data capture scenarios with different levels of viewpoint density and different camera arrangements. For the work in this paper, we will demonstrate our results on the two most common plenoptic point cloud (or SLF) capture scanarios: a multi-camera studio capture, where there is a collection of stationary cameras to capture the scene simultaneously, and video fly-by capture, where a static scene is captured by a single camera that moves in a trajectory throughout the scene. The latter is equivalent to capturing each frame by a stationary camera at a different location. For realistic capture scenarios, only a sparse collection of angular directions are sampled for each surface point, as illustrated in Fig. 1 (left) for the video fly-by scenario, and Fig. 1 (right) for the multi-camera studio scenario. In the multi-camera studio scenario, the collection of directions (θ, ϕ) from which a surface point is captured corresponds to the number of cameras that see the point. If there are 12 cameras, and a surface point is visible to 7 of these (and occluded in 5), then there are 7 directions for which there is valid data. The number of valid samples generally vary from point to point. In the video fly-by scanario, the collection of directions (θ, ϕ) from which a surface point is captured corresponds to the trajectory of the camera relative to the point. If the point appears in 30 video frames, then there are 30 directions for which there is valid data.

A. Synthetic Data HRDF Representation

The synthetically-generated SLF datasets used in our experiments are variations (i.e., with different surface roughness values) of the Bunny and Gold Sphere used in [23]. In these datasets, each surface point $s = (x, y, z)$ has values for the radiance at outgoing angles $a = (\theta_\ell, \phi_\ell)$ (where ℓ stands for "local") on a regular grid, as illustrated in the example in Fig. 2 (left) for two surface points s_1 and s_2. The HRDF

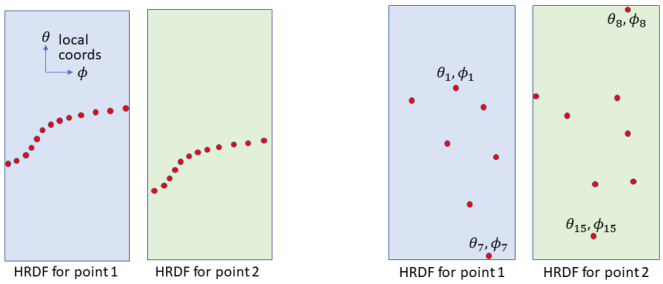

Fig. 1: For realistic data capture, examples of HRDF angular sampling patterns for two spatial points (1 and 2). Left: Data collected by video fly-by. Right: Data collected in a multi-camera studio. In an HRDF represenation, angular directions are generally different for different spatial points, as they are in a local coordinate system.

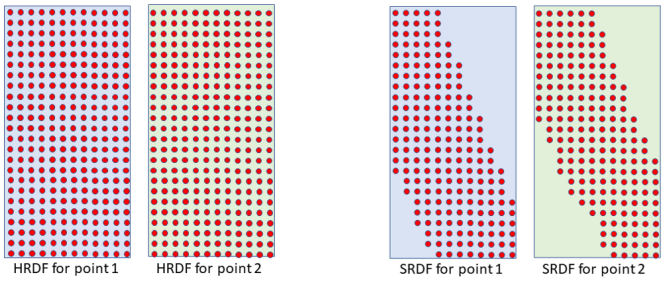

Fig. 2: Examples of HRDF angular sampling patterns for two spatial points (1 and 2). Left: Original HRDF parameterization in a local coordinate system. Right: HRDF re-parameterized into a global coordinate system, in which half the angular directions are invalid, but a different half depending on the spatial point.

angular parameters θ_ℓ and ϕ_ℓ are in a coordinate system that is local to each surface point, where $\theta_\ell \in [-\pi, \pi]$ is the azimuth, and $\phi_\ell \in [0, \pi/2]$ is the elevation above the horizon as seen from the surface point, with $\phi_\ell = \pi/2$ corresponding to the direction normal to the surface at the point. Using these local coordinates, the HRDF at each point can be represented by an image in $[-\pi, \pi] \times [0, \pi/2]$.

The local coordinates allow the HRDFs at all the points to be represented by images on the same domain, $[-\pi, \pi] \times [0, \pi/2]$, but those images may not be well correlated across surface points, because the coordinate system changes from point to point. To address this, we re-parameterize the local hemispherical coordinates of the HRDF into global spherical coordinates. Let us call the re-parameterized HRDFs, SRDFs (*Spherical Radiance Distribution Functions*). Though there are many different ways to parameterize SRDFs, in this paper we focus on a longitude-latitude global parameterization. Let us use the symbols (θ, ϕ) as the global angular coordinates, with $\theta \in [-\pi, \pi]$ being the azimuth (longitude) and $\phi \in [-\pi/2, \pi/2]$ being the elevation (latitude), with the understanding that they are *global* coordinates (shared by all surface points) rather than local. We explain the details of this local to global reparameterization in Section VI-B. Note that the conversion of the local HRDF Surface Light Field representation into a global coordinate system also allows us to simulate a real capture in a multi-camera studio environment, or a video fly-by capture (see the examples in Figs. 3 and 6).

A side effect of putting the RDF (*Radiance Distribution Function*) into global coordinates is that only about half of the angular directions are valid for each point, so if we represent a point's RDF as an image, we have to keep track of which angular directions are valid and which are invalid for each point, as illustrated in the example in Fig. 2 (right). Rather than representing the collection of SRDFs with images, and keeping track of valid and invalid data in those images, we can represent the collection of SRDFs as a list of *valid* data only, in a 5-D or 6-D point cloud, as introduced in Section III.

B. Local HRDF to Global Viewmap Reparameterization

For each surface point $s = (x, y, z)$, we sample the RDF on a regular grid of view directions, called a *viewmap*. If the viewmap has dimensions $vmap_h \times vmap_w$, then its (i, j)th view direction has global angular coordinates

$$\begin{aligned} \theta &= 2\pi[i/(vmap_h - 1)] - \pi, \\ \phi &= \pi[j/(vmap_w - 1)] - \pi/2, \end{aligned} \tag{13}$$

for $i = 0, \ldots, vmap_h - 1$ and $j = 0, \ldots, vmap_w - 1$. These global angular coordinates correspond to 3-D coordinates (x^a, y^a, z^a) on the surface of a unit sphere, according to (4). In turn, (x^a, y^a, z^a) correspond to local angular coordinates (θ_ℓ, ϕ_ℓ) at the surface point s, as follows. We use the local coordinate system at s given by the local coordinate basis $\hat{n}, \hat{t}, \hat{b}$, where $\hat{n}$ is the unit vector *normal* to the surface at s, $\hat{t}$ is a specified unit vector *tangent* to the surface at s, and $\hat{b}$ is a unit vector perpendicular to both $\hat{t}$ and $\hat{n}$, known as the *bitangent* vector. Specifically,

$$\begin{aligned} n &= (x^a, y^a, z^a) \cdot \hat{n}, \\ t &= (x^a, y^a, z^a) \cdot \hat{t}, \\ b &= (x^a, y^a, z^a) \cdot \hat{b} \end{aligned} \tag{14}$$

are the coefficients of the unit vector (x^a, y^a, z^a) in the basis $\hat{n}, \hat{t}, \hat{b}$. In turn, these coefficients map to the local angular coordinates (θ_ℓ, ϕ_ℓ) as:

$$\begin{aligned} \theta_\ell &= \arctan 2(t, b), \\ \phi_\ell &= \arcsin(n). \end{aligned} \tag{15}$$

We use the coordinates (θ_ℓ, ϕ_ℓ) to index into the HRDF to sample the radiance at surface point s. However, if $n < 0$ (or $\phi_\ell < 0$), this indicates that s is not visible (i.e., is occluded) in that view direction. In that case, we represent the radiance in that direction in the viewmap as NaN, rather than as the radiance sampled from the HRDF.

For a camera located at position $c = (x^c, y^c, z^c)$, the appearance of the surface point $s = (x, y, z)$, if it is not occluded, is given by the RDF at s in view direction $a = (x^a, y^a, z^a)$, where

$$a = (c - s)/||c - s||. \tag{16}$$

If the camera is very far away from the object, this view direction is approximately constant,

$$a \approx c/||c||, \tag{17}$$

for all points s in the point cloud. Thus, for convenience in this paper, in our experiments we will assume that all cameras are very far away from the object, and therefore that the (i, j)th view direction in the view map for point s and the (i, j)th

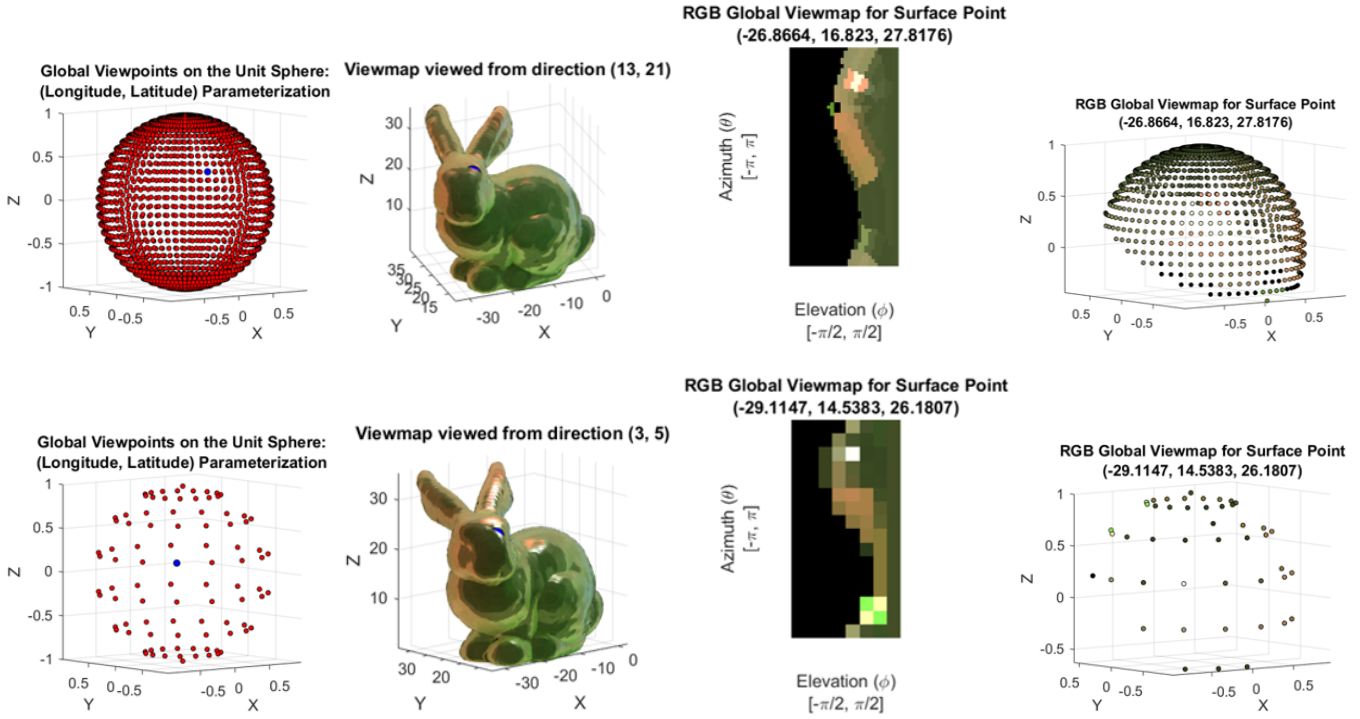

Fig. 3: SLF for the Bunny from [23], reparameterized to our global spherical viewmap representation, using a viewmap size of 64x32 (top row) and 16x8 (bottom row). The selected viewpoint and surface point are coloured in blue in the first two images in each row. The third image in each row is the global viewmap in 2-D image form, with the black areas representing viewpoint locations from which the selected surface point is not visible (these have an "NaN" colour value). The last image in each row is the 3-D version of the corresponding 2-D viewmap, and in these images only the colours for the valid viewpoints are shown.

view direction in the view map for any other point s' refer to views of the points from the same camera, or *viewpoint*. This will allow us to streamline the discussion by referring to the angular direction $a = (x^a, y^a, z^a)$, or alternatively (θ, ϕ), both as the view direction and as the camera viewpoint.[3]

Fig. 3 shows two examples of the resulting global viewmap representation, using the process described above with different chosen camera viewpoint densities on the sphere, for the Bunny dataset that was used in [23]. Fig. 4 shows that if we rotate the Bunny rendered from viewpoint (3,5) in Fig. 3 (bottom row) to a different viewpoint, the Bunny is hollow at the back, because we only record the spatial points that are *visible* from each viewpoint.

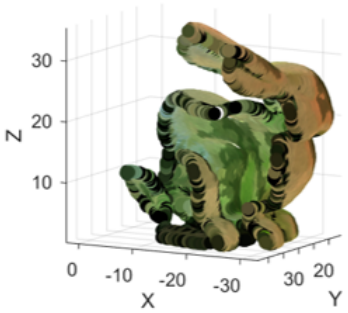

Fig. 4: Bunny from viewpoint direction (3,5) in Fig. 3 (bottom row), rotated to a different viewpoint.

VII. Experimental Procedure and Results

In this section, we present results for 5 different versions of the Bunny and 6 different versions of the Gold Sphere. Each version corresponds to a different surface roughness value (specularity is approximately 1/roughness), as shown in Fig. 5. We also consider 4 different camera viewpoint arrangements: 32x16, 16x8, 4x3, and 32x1, as shown in Fig. 6. The first 3 arrangements in Fig. 6 simulate multi-camera studio capture environments, with different levels of viewpoint density, and the 32x1 arrangement simulates a video fly-by capture.

We present results for all the Bunny and Gold Sphere datasets using a spatial rescaling range (i.e., for the (x, y, z) coordinates) of [0, 1023], obtained through the rescaling process described in Section III-B. Note that after rescaling to this range, the Bunny still has 25360 unique spatial positions, while the Gold Sphere now has 62325 unique spatial points. For all the results where we compare our 6-D methods to RAHT-KLT [4], we use RLGR entropy coding [35] for the RAHT-KLT coefficients, similarly to what is done for the 6-D RAHT and 6-D GFT coefficients (see Sections IV and V). We also use uniform scalar quantization stepsizes of [1, 2, 4, 8, 16, 32, 64], to obtain the different *rate-distortion* (R-D)

[3]This is by no means an assumption or limitation of our non-separable framework, which naturally handles the case where each spatial point may have its own angular direction to the same camera viewpoint. In contrast, separable approaches such as [4] separate the data by space along one axis and by camera along another axis. This constraint may be undesirable when the colour of a point is determined more by the view direction than by the camera identity, as would be the case for highly reflective surfaces, when the environment is reflected from each surface point according to the view direction relative to the surface normal.

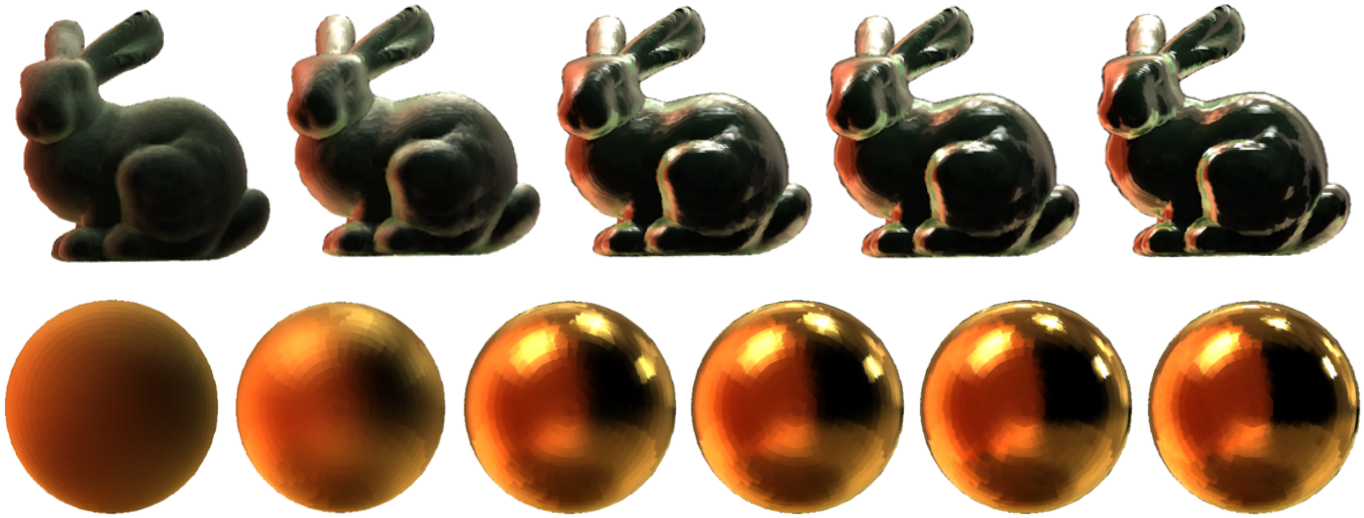

Fig. 5: Top row: Bunny with different surface roughness values (left to right): 0.5, 0.05, 0.01, 0.005, 0.001. Bottom row: Gold Sphere with different surface roughness values (left to right): 0.5, 0.05, 0.01, 0.005, 0.002, 0.001. Number of unique spatial points before rescaling: for Bunny = 25360; for Gold Sphere = 62529.

Fig. 6: Tested viewmap arrangements (left to right): 32x16, 16x8, 4x3, 32x1.

points on all the presented R-D plots. Bitrates are measured with respect to the original number of 8-D attributed points in the input datasets (see Section III), before any spatial or angular rescaling and before the removal of *invalid* points (see Section III-B). When measuring the PSNR, however, the rescaling is taken into account, as it is part of the compression methods. For 6-D RAHT and 6-D GFT, since the spatio-angular rescaling produces a smaller number of attributed points than before rescaling and the colours of any attributed points with identical Morton codes (i.e., identical spatio-angular positions) after rescaling are averaged before being compressed (see Section III-B), in the 6-D RAHT and 6-D GFT reconstructions we simply copy the same reconstructed colour value to all the original (before rescaling) spatio-angular points that end up having the same Morton code after rescaling. RAHT-KLT, on the other hand, does not depend on the angular rescaling, as it does not require the computation of 6-D Morton codes. RAHT-KLT may require only the rescaling of the spatial (x, y, z) coordinates, if they are not already in an integer range. Therefore, for RAHT-KLT, we also average the colours of all the spatio-angular points that end up being identical after rescaling of the spatial coordinates, and we pass these colour values as input to RAHT-KLT. For the RAHT-KLT reconstruction, similarly to the case for 6-D RAHT and 6-D GFT, we simply copy the same reconstructed colour value to all the original (before rescaling) spatio-angular points that end up being the same after spatial rescaling. This means that both RAHT-KLT and our 6-D methods suffer some loss due to coordinate rescaling, more so for the 6-D methods because they require a rescaling in the angular dimension as well.

Finally, since 6-D RAHT and 6-D GFT can work on the *incomplete* viewpoint data, they reconstruct only the *valid* (visible) viewpoint colours for each spatial point. RAHT-KLT works only on the *complete* viewpoint data and therefore reconstructs a colour for *every* viewpoint (visible and invisible), for each spatial point. To ensure a fair PSNR comparison, we consider only the *valid* colour values that are reconstructed by all the methods, and ignore the invalid values. The reference for PSNR computation is thus the set of original colour values, corresponding to the original attributed points before coordinate rescaling, and considering only the *valid* attributed points.

1) Impact of angular scaling: We first investigate the impact of angular rescaling, given a chosen suitable spatial rescaling range, on the rate-distortion performance of 6-D RAHT, for datasets with different surface specularities and different simulated camera arrangements. We compare the results to RAHT-KLT in each case, which is independent of the angular rescaling. Due to space limitations in the paper, we present here only the Y PSNR plots, but we have observed that the Cb and Cr plots (the input data is in YCbCr colour space) follow similar patterns. In Figs. 7 and 10, we see that when a suitable angular rescaling range is chosen for 6-D RAHT, the performance improvement over RAHT-KLT is significant. For the denser viewmap arrangements of 32x16 and 16x8, as well as for the video fly-by simulation arrangement of 32x1, 6-D RAHT is generally able to achieve between 5-10 dB better PSNR than RAHT-KLT at a similar (low) bitrate, and an even greater improvement than that at the higher bitrates. For the much sparser viewpoint arrangement of 4x3, the improvement of 6-D RAHT over RAHT-KLT is smaller, but nevertheless ranging between around 2 dB at the lowest bitrates up to around 10 dB at the highest bitrates. We also see in Fig. 7 that the 6-D RAHT performance generally continues

to improve from angular rescaling ranges of 0-0 to 0-511, and then begins to worsen. This indicates that the most suitable angular rescaling range for this dataset is 0-511 (except for the 4x3 viewpoint arrangement, where the best angular rescaling range seems to be 0-1023). For the Gold Sphere in Fig. 10, we see that the best angular rescaling range tends to be 0-1023. Figs. 7 and 10 also demonstrate that when too small an angular scaling range is used, the PSNR saturates at a low value, because the chosen angular scaling does not allow enough different viewpoints to be represented to be able to accurately reconstruct the plenoptic colour values on the highly specular object surfaces. In general, our observations from Figs. 7 and 10 indicate that for a fixed spatial scale, the angular scale trades off the spatial and angular resolutions of the 3-D object. Since for the Bunny and Gold Sphere, there is little variation in colour across the different spatial points relative to the variation in colour in the angular dimension, the optimal angular scales are quite high. Furthermore, we see that the density of viewpoints also has an effect on the optimal angular scale: for the denser viewpoint arrangements in Figs. 7 and 10, the differences in R-D performance between the different angular scales are more gradual and more evident than the differences in R-D performance for the sparser viewpoint arrangement 4x3, because the denser viewpoint arrangements allow more correlations between points in the angular dimension.

2) Impact of specularity: Figs. 9 and 12 show the results on the Bunny and Gold Sphere, respectively, with different surface roughness values, when we use the "optimal" angular scaling range found in Figs. 7 and 10 (respectively). We see that, as expected, the smaller the surface roughness in the input data (i.e., the greater the surface specularity), the more bits are required to encode the plenoptic colour vectors to achieve a similar reconstruction quality. We also see that at all of the tested surface roughness values, 6-D RAHT significantly outperforms RAHT-KLT. Furthermore, the examples in Figs. 8 and 11 demonstrate visually that 6-D RAHT is able reproduce the colours on the 3-D objects more faithfully and smoothly than RAHT-KLT at a similar bitrate. Note that the reconstructions in Figs. 8 and 11 show only the *visible* surface points, and their corresponding colours, at the selected viewpoint.

3) Comparative assessment of 6-D GFT, 6-D RAHT, and RAHT-KLT: Fig. 13 compares 6-D GFT to 6-D RAHT and RAHT-KLT, for different GFT block sizes, using the same angular rescaling range for 6-D RAHT and 6-D GFT that produced the best 6-D RAHT results for the Bunny in Fig. 7. We see that the 6-D GFT performs similarly to 6-D RAHT, but 6-D RAHT usually has the best performance. For the 6-D GFT, our experiments show that the smaller block sizes generally afford better R-D performance at the low bitrates and the larger blocksizes at the higher bitrates. Perhaps this is because a smaller block size allows local geometric properties of the point cloud to be captured better, and at the lowest bitrates where there is the coarsest quantization, this better localised capturing of the point cloud geometry helps to offset the large reconstruction error caused by the coarse quantization, resulting in a slightly higher PSNR than for a larger blocksize at a similar bitrate. In all the cases, however, there is a clear performance difference between the 6-D methods and RAHT-KLT. We have found that the R-D results presented in Fig. 13 are also representative of the R-D results when using different viewmap arrangements and different surface roughness values for the Bunny, and we have observed similar patterns for the Gold Sphere dataset.

VIII. Complexity of the Proposed Approach

For the proposed approach, some example runtimes for the key algorithm steps in the conversion of the input plenoptic point cloud into our global viewmap representation and associated 6-D space, are shown in Table I, below. The times in Table I have been taken from our prototype MATLAB implementation and are not intended to be optimal; they are simply intended to give the reader an idea of the time complexity of different key parts of the proposed algorithm.

Key Algorithm Steps	Approximate Time Taken (s)			
(Global Viewmap Arrangement)	32x16	16x8	4x3	32x1
Local HRDF to global SLF conversion	7.42	2.52	0.79	1.10
Creating *complete* and *incomplete* sets of 8-D attributed points	1.94	1.19	0.95	1.00
Spatial and angular point rescaling	2.70	0.61	0.07	0.17
Morton code comp. & related processing	13.82	3.19	0.29	0.84

TABLE I: Approximate times taken for processing the Bunny dataset from Fig. 13, in MATLAB R2018b, on a 64-bit Dell Latitude 7480 laptop with an Intel Core i7-7600U CPU (2 cores), an integrated Intel HD Graphics 620 graphics card, and a Windows 10 operating system.

We see in Table I that the most time-consuming part of the proposed algorithm for converting an input plenoptic point cloud (in HRDF representation) into our proposed 6-D framework is usually the Morton code computation and related processing (see Section III-B for the algorithmic details). While there exist a number of different methods for implementing a Morton code computation, in our current prototype this is implemented as a simple for-loop to access different bits of the input (rescaled) 6-D coordinates in order to interleave them. This means that our algorithm has approximately linear time complexity with respect to the number of input spatio-angular points, as is confirmed by the example runtimes presented in Table I. Note that for the examples in Tables I-III, the number of input *spatial* points remains constant, so it is the number of viewing directions (in the global viewmap arrangement) that causes the variation in runtimes. In terms of data (space) complexity, this is also linear with respect to the number of input spatio-angular points.

In Table II, we provide some example times for 6-D RAHT encoding and decoding on the Bunny from Fig. 13. We see that both the encoder and decoder execute very quickly (particularly the decoder) and that they are almost linear in time complexity relative to the number of input spatio-angular points.

	Approximate Time Taken (s)			
(Global Viewmap Arrangement)	32x16	16x8	4x3	32x1
6-D RAHT Encoder	16.75	3.73	0.19	1.21
6-D RAHT Decoder	4.62	0.97	0.04	0.29

TABLE II: Approximate times taken for 6-D RAHT encoding and decoding of the Bunny dataset from Fig. 13, on the same computer system as for Table I.

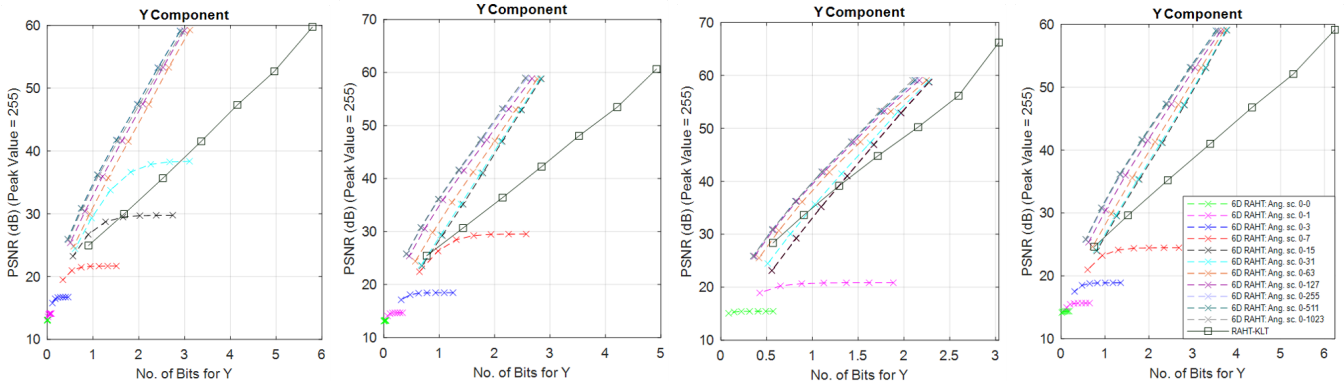

Fig. 7: R-D curves for 6-D RAHT (valid viewpoints) using different angular scales, versus RAHT-KLT (valid + invalid viewpoints), for Bunny with surface roughness 0.01 and spatial rescaling 0-1023, using global viewmap sizes (left to right): 32x16, 16x8, 4x3, 32x1.

Fig. 8: Comparing reconstructions of Bunny from Fig. 7 at viewpoint arrangement 16x8, at similar bitrates for 6-D RAHT and RAHT-KLT, using an angular rescaling of 0-511 for 6-D RAHT. Left to right: Input, 6-D RAHT reconstruction at 0.7 bpp and PSNR 30.8 dB, RAHT-KLT reconstruction at 0.8 bpp and PSNR 25.4 dB, 6-D RAHT reconstruction at 1.4 bpp and PSNR 41.6 dB, RAHT-KLT reconstruction at 1.4 bpp and PSNR 30.7 dB.

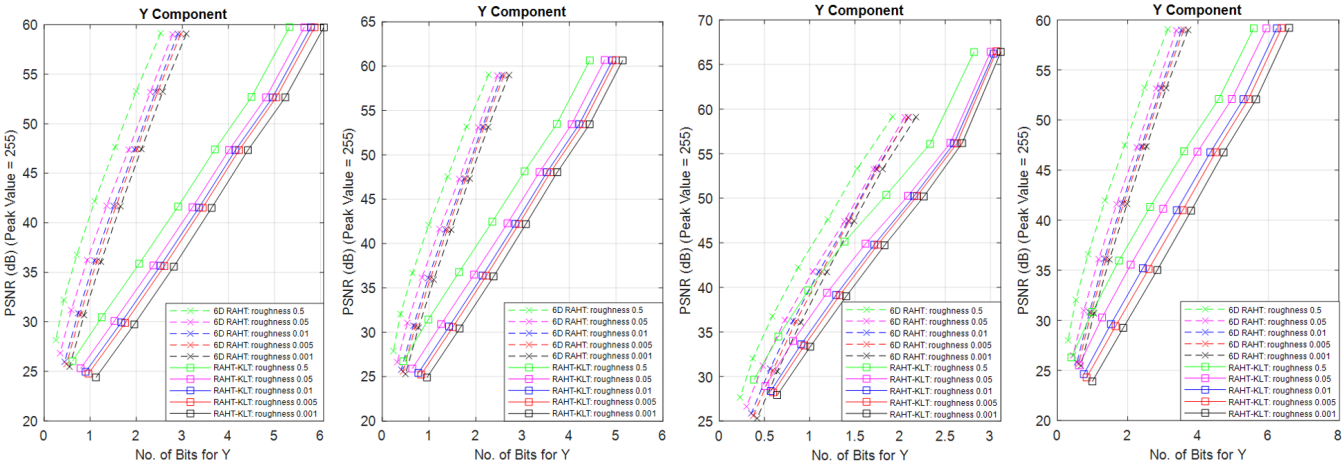

Fig. 9: R-D curves for 6-D RAHT (valid viewpoints) versus RAHT-KLT (valid + invalid viewpoints), for Bunny with different surface roughness values, spatial rescaling 0-1023, and the best angular scale for 6-D RAHT shown in Fig. 7, using global viewmap sizes (left to right): 32x16, 16x8, 4x3, 32x1. Angular scales used (left to right): 0-511, 0-511, 0-1023, 0-511.

In Table III, we present some example runtimes for key steps in our proposed 6-D GFT algorithm, again for the Bunny from Fig. 13. We see in Table III that the most time-consuming operations by far are the graph construction and eigenvector/eigenvalue computations across all the 6-D blocks, and that these times increase approximately linearly with the number of spatio-angular points. In fact, the graph construction is what consumes most of the time here, rather than the eigenvector/eigenvalue computations, which are usually very fast (almost negligible, or a few seconds per block at the most for the tests in this paper). The main reasons why the graph construction is so time-consuming are that: (i) in our current prototype implementation, we have a sequential processing for the blocks, which can be very time-consuming when there is a large number of blocks to process, and (ii) we use a fully weighted graph, where each node is connected to every other node within a block (see Section V-B), so we must compute the distance between each node and every other node in the same block. In our current implementation, we have reduced the time complexity of the latter operation from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$ (where n is the number of spatio-angular points in a block), by using highly vectorised code. We also

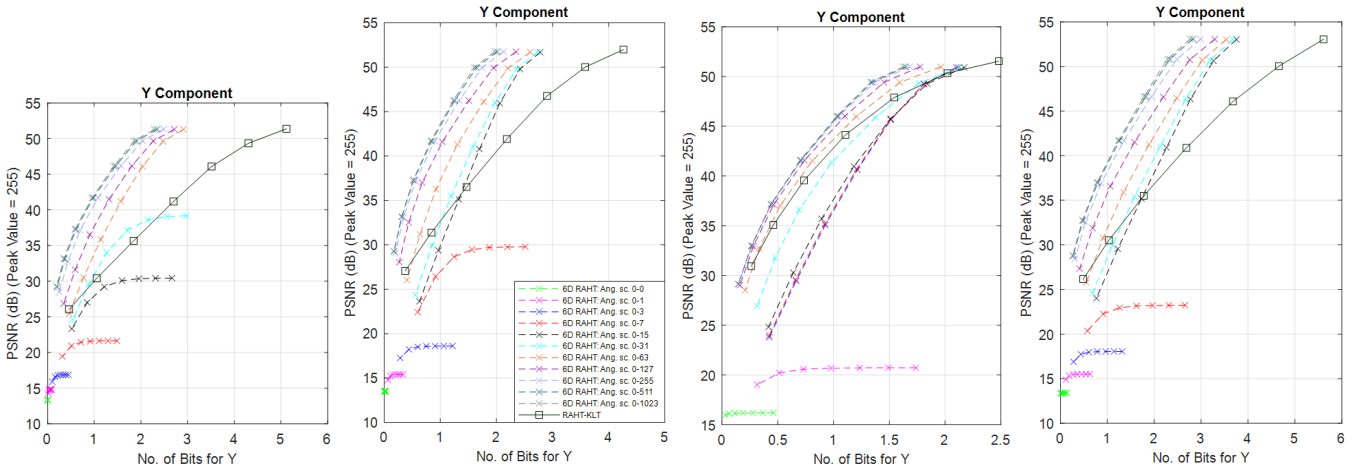

Fig. 10: R-D curves for 6-D RAHT (valid viewpoints) using different angular scales, versus RAHT-KLT (valid + invalid viewpoints), for Gold Sphere with surface roughness 0.002 and spatial rescaling 0-1023, using global viewmap sizes (left to right): 32x16, 16x8, 4x3, 32x1.

Fig. 11: Comparing reconstructions of Gold Sphere from Fig. 10 at viewpoint arrangement 32x16, at similar bitrates for 6-D RAHT and RAHT-KLT, using an angular rescaling of 0-1023 for 6-D RAHT. Left to right: Input, 6-D RAHT reconstruction at 0.4 bpp and PSNR 33.2 dB, RAHT-KLT reconstruction at 0.5 bpp and PSNR 26.1 dB, 6-D RAHT reconstruction at 1.0 bpp and PSNR 41.7 dB, RAHT-KLT reconstruction at 1.1 bpp and PSNR 30.4 dB.

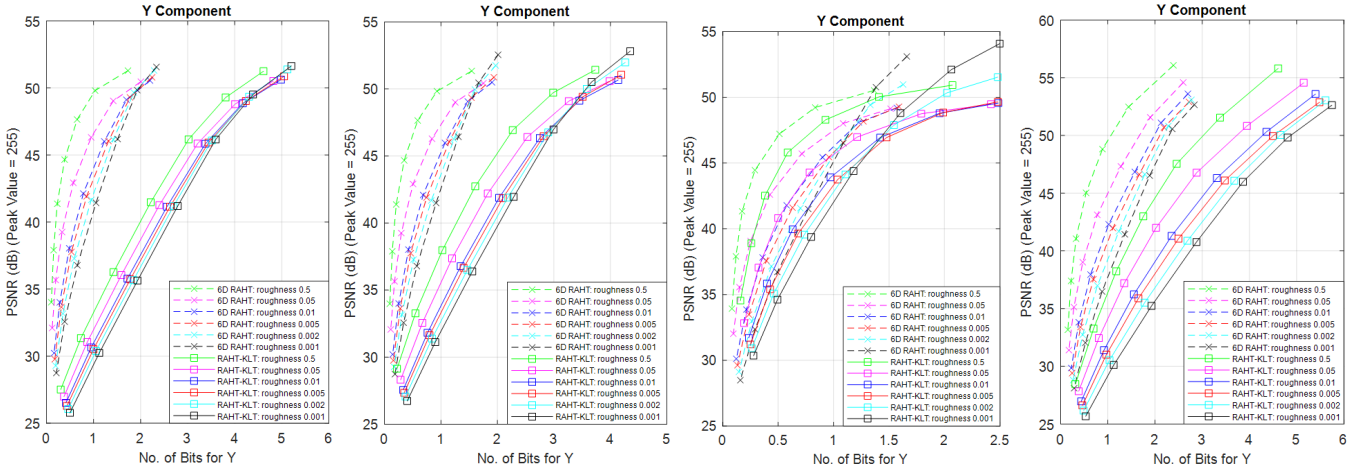

Fig. 12: R-D curves for 6-D RAHT (valid viewpoints) versus RAHT-KLT (valid + invalid viewpoints), for Gold Sphere with different surface roughness values, spatial rescaling 0-1023, and the best angular scale for 6-D RAHT shown in Fig. 10, using global viewmap sizes (left to right): 32x16, 16x8, 4x3, 32x1. The same angular scale (0-1023) was used for all the 6-D RAHT graphs in this figure.

believe that the total runtime for the graph construction across all the blocks could be dramatically improved by having a parallel-processing implementation that is optimised for dealing with a large number of blocks. In fact, we see in Table III that the smaller block sizes lead to significantly larger total processing times for the graph construction than the larger block sizes, simply because when the block sizes are smaller there is a significantly larger number of them to process. The latter observation generally also applies to the colour analysis (decomposition on the eigenvector basis) at the encoder and the colour synthesis (reconstruction) operations at the decoder, which are currently applied sequentially across all the blocks. In fact, both the analysis and synthesis per block are very fast (negligible time taken), but for the results in Table III we decided to trade off time complexity and data complexity: instead of storing the set of eigenvectors for *all*

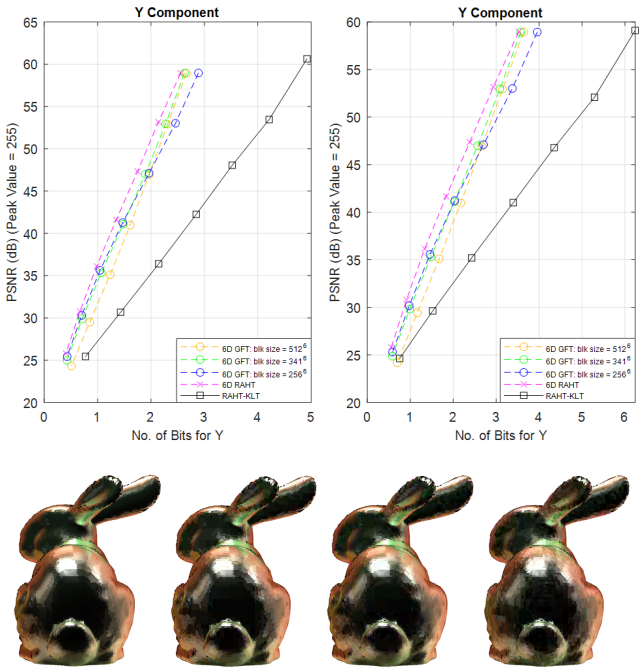

Fig. 13: (Top row) R-D curves for 6-D GFT (valid viewpoints) versus 6-D RAHT (valid viewpoints) versus RAHT-KLT (valid + invalid viewpoints), for Bunny with surface roughness 0.01, spatial rescaling 0-1023, and angular rescaling 0-511 (for 6-D RAHT and 6-D GFT), using different blocksizes for the GFT, and viewpoint arrangement 16x8 (left) and 32x1 (right). (Bottom row) Bunny reconstructions from viewpoint arrangement 32x1, showing (left to right): Input, 6-D RAHT reconstruction at 0.9 bpp and PSNR 30.8 dB, 6-D GFT reconstruction using block size 256^6 at 1.0 bpp and PSNR 30.2 dB, RAHT-KLT reconstruction at 0.8 bpp and PSNR 24.6 dB.

the 6-D blocks in memory, we save them to disk and then load the corresponding set of eigenvectors per block when we process that block, both at the encoder and decoder ends. The loading of the eigenvectors normally takes around 1 s per block (or very slightly longer for larger blocks), but again since these operations are currently sequential, the time adds up over many blocks. All of the runtimes shown in Table III should therefore be considered exemplary prototypes only, as there is significant room for improvement in terms of speed optimisations.

Based on the results presented in this paper, as well as the complexity analysis in the current section, it seems that between 6-D RAHT and 6-D GFT, using 6-D RAHT may be the best option to achieve both, better rate-distortion performance (e.g., see Fig. 13) and faster speed of codec execution.

6-D GFT Key Algorithm Steps	Approximate Time Taken (All Blocks) (s)			
(Global Viewmap Arrangement)	32x16	16x8	4x3	32x1
6-D block subdivision [blk size = 341^6]	0.58	0.12	0.01	0.04
6-D block subdivision [blk size = 256^6]	0.57	0.14	0.01	0.05
Graph construction + eig. comp. [blk size = 341^6]	7601.12	669.22	56.93	186.68
Graph construction + eig. comp. [blk size = 256^6]	15143.79	3849.02	284.85	963.35
Colour analysis and encoding [blk size = 341^6]	1657.98	357.91	22.42	108.25
Colour analysis and encoding [blk size = 256^6]	8709.55	1119.98	74.19	283.64
Colour decoding and synthesis [blk size = 341^6]	787.64	96.15	5.41	15.69
Colour decoding and synthesis [blk size = 256^6]	563.53	149.66	14.47	50.03

TABLE III: Approximate times taken for key steps of the 6-D GFT encoding and decoding of the Bunny dataset from Fig. 13, on the same computer system as for Tables I and II.

IX. Conclusion

In this paper, we introduced a 6-D representation for the joint (non-separable) spatio-angular compression of the colour attributes of plenoptic point clouds. Our representation considers a plenoptic point cloud as a set of spatio-angular locations (x, y, z, x^a, y^a, z^a) with associated colour values, where only the "valid" colour data (i.e., only for the spatial positions that are *visible* from each viewpoint) is represented and encoded, rather than encoding the colours for *all* the viewpoints as is done by the current state-of-the-art plenoptic colour coding method, RAHT-KLT [4]. We also proposed extensions of the well-known RAHT [14, 15] and GFT compression methods to our 6-D spatio angular space. These 6-D methods have been tested on synthetic SLF data with various degrees of surface specularity, illuminated with different-coloured light sources, and captured with different simulated camera arrangements and different viewpoint densities. In all these scenarios, we have demonstrated significant rate-distortion improvements over the state-of-the-art. As a next step for future work, we would like to investigate how non-synthetic plenoptic datasets, such as the 8iVSLF data [2], can be placed into our 6-D framework and compressed using the proposed 6-D methods.

X. Acknowledgements

This work has been funded by the EU H2020 Research and Innovation Programme, grant no. 694122 (ERC advanced grant CLIM). The authors would also like to thank Gustavo Sandri and Ricardo de Queiroz for the RAHT-KLT code, and the reviewers of this paper for their valuable feedback.

References

[1] S. J. Koppal. "Lambertian Reflectance". In: *Computer Vision: A Reference Guide*. Ed. by K. Ikeuchi. 2014, pp. 441–443.

[2] M. Krivokuća, P. A. Chou, and P. Savill. *8i Voxelized Surface Light Field (8iVSLF) Dataset*. input document m42914. Ljubljana, Slovenia: ISO/IEC JTC1/SC29 WG11 (MPEG), Ljubljana, Slovenia, July 2018.

[3] G. Sandri, R. de Queiroz, and P. A. Chou. "Compression of Plenoptic Point Clouds Using the Region-Adaptive Hierarchical Transform". In: *25th IEEE Int. Conf. on Image Processing (ICIP)*. Oct. 2018, pp. 1153–1157.

[4] G. Sandri, R. L. de Queiroz, and P. A. Chou. "Compression of Plenoptic Point Clouds". In: *IEEE Trans. on Image Processing* 28.3 (2019), pp. 1419–1427.

[5] X. Zhang et al. "A Framework for Surface Light Field Compression". In: *IEEE Int. Conf. on Image Processing (ICIP)*. 2018, pp. 2595–2599.

[6] X. Zhang et al. "Surface Light Field Compression Using a Point Cloud Codec". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9.1 (Mar. 2019), pp. 163–176.

[7] E. H. Adelson and J. R. Bergen. "The Plenoptic Function and the Elements of Early Vision". In: *Computational Models of Visual Processing*. 1991, pp. 3–20.

[8] C. Cao, M. Preda, and T. Zaharia. "3D Point Cloud Compression: A Survey". In: *Int. Conf. on 3D Web Technology*. Web3D '19. LA, CA, USA, 2019, pp. 1–9.

[9] S. Schwarz et al. "Emerging MPEG Standards for Point Cloud Compression". In: *IEEE J. on Emerging and Selected Topics in Circuits and Systems* 9.1 (Mar. 2019), pp. 133–148.

[10] D. Graziosi et al. "An Overview of Ongoing Point Cloud Compression Standardization Activities: Video-based (V-PCC) and Geometry-based (G-PCC)". In: *APSIPA Trans. on Signal and Information Processing* 9 (Apr. 2020).

[11] T. Ebrahimi et al. "JPEG Pleno: Toward an Efficient Representation of Visual Reality". In: *IEEE MultiMedia* 23.4 (Oct. 2016), pp. 14–20.

[12] D. Naik and S. Schwarz. "Surface Light Field Coding for Dynamic 3D Point Clouds". In: *European light field imaging workshop-2019*. 2019.

[13] L. Li et al. "Video-Based Compression for Plenoptic Point Clouds". In: *Data Compression Conf. (DCC)*. 2020, pp. 378–378.

[14] R. L. de Queiroz and P. A. Chou. "Compression of 3D Point Clouds Using a Region-Adaptive Hierarchical Transform". In: *IEEE Trans. on Image Processing* 25.8 (Aug. 2016), pp. 3947–3956.

[15] G. Sandri, R. L. de Queiroz, and P. A. Chou. *Comments on "Compression of 3D Point Clouds Using a Region-Adaptive Hierarchical Transform"*. 2018. arXiv: 1805.09146 [eess.IV].

[16] D. I. Shuman et al. "The Emerging Field of Signal Processing on Graphs: Extending High-Dimensional Data Analysis to Networks and Other Irregular Domains". In: *IEEE Signal Process. Mag.* 30.3 (May 2013), pp. 83–98.

[17] A. Ortega et al. "Graph Signal Processing: Overview, Challenges, and Applications". In: *Proceedings of the IEEE* 106.5 (2018), pp. 808–828. DOI: 10.1109/JPROC.2018.2820126.

[18] C. Zhang, D. Florêncio, and C. Loop. "Point Cloud Attribute Compression with Graph Transform". In: *IEEE Int. Conf. on Image Processing (ICIP)*. 2014, pp. 2066–2070.

[19] E. Pavez et al. "Region Adaptive Graph Fourier Transform for 3D Point Clouds". In: *IEEE Int. Conf. on Image Processing (ICIP)*. 2020, pp. 2726–2730.

[20] M. Krivokuća and C. Guillemot. "Colour Compression of Plenoptic Point Clouds Using RAHT-KLT with Prior Colour Clustering and Specular/Diffuse Component Separation". In: *IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pp. 1978–1982.

[21] E. J. Candès et al. "Robust Principal Component Analysis?" In: *Journal of the ACM* 58.3 (May 2011), 11:1–11:37.

[22] I. Ihrke, J. Restrepo, and L. Mignard-Debise. "Principles of Light Field Imaging: Briefly revisiting 25 years of research". In: *IEEE Signal Processing Magazine* 33.5 (Sept. 2016), pp. 59–69.

[23] E. Miandji, J. Kronander, and J. Unger. "Learning Based Compression of Surface Light Fields for Real-Time Rendering of Global Illumination Scenes". In: *SIGGRAPH Asia, Technical Briefs*. 2013.

[24] G. Miller, S. Rubin, and D. Ponceleón. "Lazy Decompression of Surface Light Fields for Precomputed Global Illumination". In: *Rendering Techniques*. Springer, Vienna, 1998, pp. 281–292.

[25] D. N. Wood et al. "Surface Light Fields for 3D Photography". In: *27th Annual Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. July 2000, pp. 287–296.

[26] W.-C. Chen et al. "Light Field Mapping: Efficient Representation and Hardware Rendering of Surface Light Fields". In: *ACM Trans. on Graphics* 21.3 (July 2002), pp. 447–456.

[27] D. Naik et al. "Surface Lightfield Support in Video-based Point Cloud Coding". In: *IEEE Int. Workshop on Multimedia Signal Processing (MMSP)*. 2020, pp. 1–6.

[28] M. M. Hannuksela et al. "Overview of the Multiview High Efficiency Video Coding (MV-HEVC) Standard". In: *IEEE Int. Conf. on Image Processing (ICIP)*. 2015, pp. 2154–2158.

[29] M. Bader. *Space-Filling Curves: An Introduction With Applications in Scientific Computing*. Texts in Computational Science and Engineering. Springer Berlin Heidelberg, 2012.

[30] G. P. Sandri et al. "Integer Alternative for the Region-Adaptive Hierarchical Transform". In: *IEEE Signal Processing Letters* 26.9 (2019), pp. 1369–1372.

[31] S. Lassere and D. Flynn. *On an Improvement of RAHT to Exploit Attribute Correlation*. input document m47378. Geneva: ISO/IEC JTC1/SC29/WG11 MPEG, Mar. 2019.

[32] A. L. Souto and R. L. de Queiroz. "On Predictive RAHT For Dynamic Point Cloud Coding". In: *2020 IEEE International Conference on Image Processing (ICIP)*. 2020, pp. 2701–2705.

[33] E. Pavez et al. "Multi-resolution Intra-predictive Coding of 3D Point Cloud Attributes". In: *2021 IEEE International Conference on Image Processing (ICIP)*. 2021.

[34] P. A. Chou, M. Koroteev, and M. Krivokuća. "A Volumetric Approach to Point Cloud Compression—Part I: Attribute Compression". In: *IEEE Transactions on Image Processing* 29 (2020), pp. 2203–2216.

[35] H.S. Malvar. "Adaptive Run-length/Golomb-Rice Encoding of Quantized Generalized Gaussian Sources with Unknown Statistics". In: *Data Compression Conference (DCC'06)*. 2006, pp. 23–32.

[36] G. Taubin. "A Signal Processing Approach to Fair Surface Design". In: *22nd Annual Conf. on Computer Graphics and Interactive Techniques*. SIGGRAPH '95. 1995, pp. 351–358.

[37] Z. Karni and C. Gotsman. "Spectral Compression of Mesh Geometry". In: *27th Annual Conf. on Computer Graphics and Interactive Techniques*. SIGGRAPH. 2000, pp. 279–286.

Maja Krivokuća received her Bachelor of Engineering degree (First Class Honours) in Computer Systems Engineering from the University of Auckland, New Zealand, in 2010, and her PhD from the same university in 2015. The focus of Maja's PhD 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 at Mitsubishi Electric Research Laboratories (Cambridge, Massachusetts, USA), researching new methods for compressing large-scale 3D point clouds obtained from Mobile Mapping Systems, at 8i (Wellington, New Zealand), researching and developing new shape compression algorithms for 3D point clouds in virtual/augmented reality applications, and at INRIA (Rennes, France), researching new compression algorithms for plenoptic point clouds as part of the SIROCCO research team and the Computational Light Fields Imaging project. Maja is currently at InterDigital (Rennes, France), working on future 3D mesh and point cloud coding standards. Maja was also actively involved in contributing to the emerging MPEG Point Cloud Compression standards (notably G-PCC). Maja's main research interests are in the areas of data compression and information theory, as well as in signal and image processing.

Ehsan Miandji received his B.Sc. degree from Azad University, Tehran, Iran in 2008, and his M.Sc. and Ph.D. in computer graphics and image processing from Linköping University, Sweden, in 2012 and 2018, respectively. He pursued post-doctoral research at INRIA, Rennes, France in 2020. Currently, he is a post-doctoral research associate at the Department of Science and Technology, Linköping University. Dr. Miandji has published peer-reviewed journal papers in compressive light field photography, compression and compressed sensing of visual data, photorealistic image synthesis, as well as theoretical work on compressed sensing. He was an IPC member for Eurographics short papers program in 2020 and 2021. He is a member of IEEE, ACM SIGGRAPH, and Eurographics.

Christine Guillemot is an IEEE fellow and Director of Research at INRIA. She holds a Ph.D. degree from ENST (Ecole Nationale Superieure des Telecommunications) Paris, and an Habilitation for Research Direction from the University of Rennes. From 1985 to Oct. 1997, she was with FRANCE TELECOM, where she was involved in various projects in the area of image and video coding and processing for TV, HDTV and multimedia. From Jan. 1990 to mid. 1991, she worked at Bellcore, NJ, USA, as a visiting scientist. Her research interests are signal and image processing, and computer vision. She has served as Associate Editor for IEEE Transactions on Image Processing (from 2000-2003, and from 2014-2016), for IEEE Transactions on Circuits and Systems for Video Technology (from 2004-2006), and for IEEE Transactions on Signal Processing (2007-2009). She has served as senior member of the editorial board of the IEEE Journal on Selected Topics in Signal Processing (2013-2015) and has been senior area editor of IEEE Transactions on Image Processing (2016-2020).

Philip A. Chou received the BSE, MS, and PhD degrees respectively from Princeton University, the University of California, Berkeley, and 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 been Associate and/or Guest Editor of the IEEE Transactions on Information Theory, the IEEE Transactions on Image Processing, the IEEE Transactions on Multimedia, and IEEE Signal Processing Magazine. He has been on the organising committees of numerous conferences and workshops, including ICASSP, ICIP, and ICME. He has served on the Board of Governors as well as Technical Committees of the IEEE Signal Processing Society. He has received best paper awards from the IEEE Transactions on Signal Processing and the IEEE Transactions on Multimedia, as well as from several conferences. He is co-editor of a book on multimedia communication. He has served on IEEE Fellow evaluation committees for the IEEE Signal Processing and Computer societies, and is a Fellow of the IEEE.