TESE DE DOUTORADO

COMPRESSION OF POINT CLOUDS ATTRIBUTES

Gustavo Luiz Sandri

Brasília, Dezembro de 2019

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

CODIFICAÇÃO DE ATRIBUTOS DE NUVENS DE PONTOS

GUSTAVO LUIZ SANDRI

TESE DE DOUTORADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR.

APROVADA POR:

RICARDO LOPES DE QUEIROZ, Dr., ENE/UNB
(PRESIDENTE DA COMISSÃO)

MYLENE CHRISTINE QUEIROZ DE FARIAS, Dra., ENE/UNB
(EXAMINADOR INTERNO)

TIAGO ALVES DA FONSECA, Dr., FGA/UNB
(EXAMINADOR INTERNO)

CARLA LIBERAL PAGLIARI, Dra., IME
(EXAMINADOR EXTERNO)

Brasília, 16 de dezembro de 2019.

ABSTRACT

Point clouds have gained the attention of the community for real-time applications such as telepresence and autonomous navigation. The Region-Adaptive Hierarchical Transform has been introduced as a low complexity attribute encoding algorithm for the compression of point clouds. It presents a performance competitive with state-of-the-art algorithms.

In the original work, the Region-Adaptive Hierarchical Transform uses an Arithmetic Coder to entropy encode the quantized coefficients. We have observed that the performance of the entropy coder can be improved, without trading for complexity, by sorting the coefficients and using the Run-Length Golomb-Rice encoder, an adaptive algorithm that adapts its parameters during the encoding process. Four sorting criteria were proposed, and we observed that the breadth-first search obtained the best performance. The proposed ordering sequence largely increases the runs of zeros and introduces a smooth decay in the pattern of the coefficients, thus improving Run-Length Golomb-Rice performance.

The Moving Picture Experts Group and the Joint Photographic Experts Group have initiated activities towards the standardization of point cloud compression technologies. For the sake of reproducibility, it is preferred to use algorithms based on fixed-point arithmetic instead of floating-point arithmetic. Therefore, we also propose a modification of the Region-Adaptive Hierarchical Transform, translating the operations to fixed-point arithmetic. The modification in the transform showed little divergence in terms of rate and performance with the floating-point arithmetic one when the precision was 8 bits or above. Both algorithms are virtually identical when the precision is 10 or above.

For practical reasons, each point in the point cloud is usually associated with one single color along with other attributes. The Region-Adaptive Hierarchical Transform coder has been proposed for single-color point clouds. This approach may not be very realistic since, in real-world objects, the reflected light may significantly change with the viewing angle, especially if specular surfaces are present. For that, we are interested in a more complete representation, the plenoptic point cloud, wherein every point has associated colors in all directions. Here, we propose a compression method for such a representation. Instead of encoding a continuous function, since there is only a finite number of cameras, it makes sense to compress as many colors per voxel as cameras and to leave any intermediary color rendering interpolation to the decoder. Hence, each voxel is associated with a vector of color values for each color component. We have here developed and evaluated four methods to expand the Region-Adaptive Hierarchical Transform coder to encompass the multiple colors case. Experiments with synthetic data helped us to correlate specularity with the compression, since object specularity, at a given point in space, directly affects color disparity among the cameras, impacting the coder performance. Simulations were carried out using natural (captured) data, and results are presented as rate-distortion curves that show that a combination of Kahunen-Loève transform and the Region-Adaptive Hierarchical Transform achieves

the best performance.

In image compression, it may be possible that certain parts of an image are of higher importance than the rest of the image. Thus, we have also introduced region-of-interest coding for point clouds. The quality of the region-of-interest is prioritized in detriment to the rest of the image. The advantages of doing so are:

- The user may be more sensitive to artifacts introduced in the region-of-interest than anywhere else. Therefore, prioritizing the quality of the region-of-interest may lead to better subjective quality;
- We can reduce the number of bits necessary to encode an image by reducing the number of bits spent to encode regions that are of little interest without significantly compromising the subjective quality.

We have proposed to include region-of-interest coding in point clouds by modifying the transform. We have developed a face detection algorithm for point clouds as an example case since humans' attention tends to go toward the face, but the proposed algorithm is transparent to how the region-of-interest was defined. We have not performed subjective tests to evaluate our proposal, but have analyzed the performance in terms of rate-distortion. We observed that we can improve the quality of the region-of-interest with little impact in the overall point cloud peak signal to noise ratio.

RESUMO

As nuvens de pontos ganharam a atenção da comunidade para aplicações em tempo real, como telepresença e navegação autônoma. A *Region-Adaptive Hierarchical Transform* foi introduzida como um algoritmo de baixa complexidade para a compressão de nuvens de pontos. Ela apresenta um desempenho competitivo com algoritmos de última geração.

No trabalho original, a *Region-Adaptive Hierarchical Transform* utiliza um codificador aritmético para codificar entropicamente os coeficientes quantizados. Observamos que o desempenho do codificador de entropia pode ser melhorado, sem prejudicar a complexidade, através da ordenação dos coeficientes e usando o codificador *Run-Length Golomb-Rice*, um algoritmo adaptativo que adapta seus parâmetros durante o processo de codificação. Foram propostos quatro critérios de ordenação de coeficientes, e observamos que a busca em largura obteve o melhor desempenho. A ordenação proposta aumenta amplamente as sequências de zeros e introduz um decaimento suave no padrão dos coeficientes, melhorando a performance do *Run-Length Golomb-Rice*.

O Moving Picture Experts Group e o Joint Photographic Experts Group iniciaram atividades visando a padronização de tecnologias relacionas à compressão de nuvem de pontos. Com o objetivo de preservar compatibilidade, é preferível algoritmos baseado em aritmética de ponto fixo ao invés de aritmética de ponto flutuante. Portanto, também propomos uma modificação da *Region-Adaptive Hierarchical Transform*, traduzindo suas operações para ponto fixo. A modificação na transformada mostrou pouca divergência em termos de taxa e performance com o algoritmo que utiliza ponto flutuante quando a precisão está em 8 bits ou mais. Ambos os algoritmos são virtualmente idênticos quando a precisão é de 10 bits ou mais.

Por motivos práticos, a cada ponto da nuvem de pontos está associado uma única cor junto de outros atributos. O codificador *Region-Adaptive Hierarchical Transform* foi proposto para point clouds de uma única cor. Este método pode não ser muito realista pois, em objetos reais, a luz refletida pode variar significativamente com o ângulo de vista, especialmente se superfícies especulares estão presentes. Por isso, estamos interessados em uma representação mais completa, a plenoptic point cloud, onde todos os pontos estão associados com as cores em todas as direções. Aqui, propomos um método de compressão para esta representação. Em vez de codificar uma função contínua visto que há apenas um número finito de câmeras, faz sentido comprimir tanto números de cores quanto o número de câmeras e deixar a etapa de interpolação de cor para o decodificador, durante a renderização.

Assim, cada voxel está associado a um vetor de valores de cor. Nós desenvolvemos e avaliamos aqui 4 métodos que estendem a *Region-Adaptive Hierarchical Transform* para englobar o caso multi-cor. Os experimentos com dados sintéticos nos ajudaram a correlacionar especularidade com a compressão, já que a especularidade de um objeto, em um dado ponto no espaço, afeta diretamente a diferença de cor entra as câmeras, impactando no desempenho do codificador. Simulações foram executadas usado dados naturais (capturados) e os resultados são apresenta-

dos como curvas de taxa versus distorção que mostram que a combinação da transformada de Kahunen-Loève e a *Region-Adaptive Hierarchical Transform* atinge a melhor performance.

Na compressão de imagem, pode ser possível que certas partes de uma imagem sejam de maior importância que o restante dela. Assim, também introduzimos a codificação por região de interesse para nuvens de pontos. A qualidade da região de interesse é priorizada em detrimento do restante da imagem. As vantagens de fazer isso são:

- O usuário pode ser mais sensível aos artefatos introduzidos na região de interesse do que em qualquer outro lugar. Portanto, priorizar a qualidade da região de interesse pode levar a uma melhor qualidade subjetiva;
- Podemos reduzir o número de bits necessários para codificar uma imagem, reduzindo o número de bits gastos para codificar regiões que são de pouco interesse, sem comprometer significativamente a qualidade subjetiva.

Propusemos incluir a codificação de região de interesse em nuvens de pontos modificando a transformada. Nós desenvolvemos um algoritmo de detecção de face para nuvens de pontos como um exemplo de caso, uma vez que a atenção dos seres humanos tende a ir para a face, mas o algoritmo proposto é transparente à forma como a região de interesse foi definida. Não realizamos testes subjetivos para avaliar nossa proposta, mas analisamos o desempenho em termos de taxa-distorção. Observamos que podemos melhorar a qualidade da região de interesse com pouco impacto na relação entre sinal de pico e ruído de nuvem de pontos.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

Symbols and definitions

AC	As in Alternating Current, represents the signal component of frequency different than zero
DC	As in Direct Current, represents the signal component of zero frequency
dynamic point clouds	A sequence of point cloud frames representing how a scene develops over time
pixel	Picture element. The fundamental element of a digital image
PLY	Extension of the Polygon File Format, also known as the Stanford Triangle Format
voxel	Volume element. The fundamental element of a voxelized point cloud
voxelization	The process that convert some 3D information to a voxelized point cloud
voxelized	That has been converted to voxels

Acronyms

1D	1-dimensional
2D	2-dimensional
3D	3-dimensional
ASCII	American Standard Code for Information Interchange
CIE	Commission Internationale de l'Eclairage
HEVC	High Efficiency Video Coding, also known as H.265
JPEG	Joint Photographic Experts Group
KLT	Karhunen-Loève Transform
MPEG	Moving Picture Expert Group
PC	Point Cloud
PSNR	Peak Signal-to-Noise Ratio
ROI	Region of Interest
VPC	Voxelized Point Cloud

1 INTRODUCTION

Our ability to capture the world around us has been facilitated in the last two decades with the advancement in hardware and software [1, 2]. A scene, or an object, can be digitized using technologies ranging from active methods such as optical laser-based range scanners, structured light scanners, and LiDAR scanners to passive methods such as multi-view stereo. Such data can also be computer-generated from 3D models.

Point clouds are a typical representation of the three-dimensional world. Nevertheless, they demand a large number of resources. A typical data may contain millions of points, and a major challenge is to transmit and store high-quality point clouds efficiently. Not only is the compression necessary, but it also has to maintain a fast read and write access, and be transparent for users. Moreover, to facilitate inter-operation between production and consumption, compression standards for point clouds are being devised by The Moving Picture Experts Group (MPEG) and the Joint Photographic Experts Group (JPEG) [3, 4]. One of these technologies is called Point Cloud Compression (PCC), which is expected to be partially delivered as an ISO standard at the beginning of 2020 [5].

Current applications have made use in all fields of data-driven science. A more recent trend has seen the massive proliferation of point clouds from inexpensive commodity real-time scanners such as the Microsoft Kinect. Point cloud has impacted varied fields, including:

- **3D telepresence systems:** telepresence is a technology that allows for one person to feel and interact with someone located in another place [6]. One application is in videoconferencing. One user position, movements, and sounds are captured. This information is coded and sent to another location for display. Applications may be bidirectional, *i.e.*, both ends of the communication are being captured at the same time. Telepresence can be used for real-time communication. In this case, algorithms processing the information must be fast enough to avoid latencies. It can also be used for no real-time applications where the information is prerecorded and stored to be later played. Point clouds are one of the technologies that can be employed to achieve telepresence;
- **Autonomous navigation:** it is a system that allows for vehicles or robots to navigate an environment with little or no assistance from a human [7, 8]. Autonomous vehicles are also called unmanned vehicles. The system uses cameras or lasers to sense the environment. These sensors detect obstacles and map the neighborhood of the vehicle. The environment is represented in the form of 3D data points, a point cloud. This point cloud can be sent to a server that combines the information of different vehicles and produces more detailed maps or maps with fewer errors;
- **Virtual reality:** virtual reality creates an experience where the user is completely submerged in a simulated world. Virtual reality headsets project the simulated scene in the

user's eyes, while motion sensors capture the user movements that are transferred to the scene in order to create a more realistic experience. Applications of virtual reality range from entertainment to educational purposes [9–11]. Point clouds are a representation of 3D objects that allows free viewpoint and can be used in virtual reality applications;

- **Augmented or mixed reality:** in a very similar way to virtual reality, augmented and mixed reality also creates simulated objects that the user can feel as if they were right in front of it, but this simulation is integrated with reality, using the real world as support.
- **Modeling:** Point clouds can be used to create 3D models of the real world to be used in films, video games, landscape simulations, or city planning [12]. The 3D structure of the point cloud allows for easier handling of the data.

The compression of point clouds is typically divided into two tasks: the compression of the point's geometry and the compression of the point's attributes, which typically is its color. In this work, we focus on the compression of point clouds attributes, especially its color information.

1.1 GOALS

In this work, we aim to develop compression algorithms to encode the attributes of point clouds. Our focus is on telepresence applications where data need to be encoded and decoded in real-time with low latency. Thus, the complexity of the algorithms needs to be low enough to allow real-time applications.

Our goal is to create fast algorithms capable of dealing with real-time applications, providing competitive rate-distortion characteristics. We aim to provide a reproductive algorithm to be submitted to the MPEG standardization, capable of providing a realistic representation of the real world, taking into account the specularity of objects in a scene. We also aim to provide the capability of encoding regions-of-interest in point clouds.

1.2 PRESENTATION OF THE MANUSCRIPT

Chapter 2 presents a brief literature review. Section 2.2 introduces the main concept of point clouds, sections 2.3 and 2.4 present some algorithms for the compression of point clouds and section 2.5 introduces the plenoptic function.

Chapters 3 and 4 present two improvements made on the Region-Adaptive Hierarchical transform, a method to compress the color information of point clouds. In chapters 3 we propose a modification of the entropy coder and in Chapter 4 we modify the transform, substituting floating-point arithmetic for fixed-point arithmetic.

Chapter 5 proposes an algorithm for the compression of plenoptic point clouds. The plenoptic

point cloud is a more realistic way to represent real-world objects because it also considers that objects may change color depending on the angle they are viewed, thus better representing reflexive surfaces. The algorithm is explained in Sections 5.1 and 5.3. Sections 5.4 and 5.5 present experimental results and a conclusion, respectively.

Chapter 6 introduces our proposal for the compression of point clouds taking into account regions of interest, where we expend more bits to encode those voxels marked as ROI.

Finally, a general conclusion is given in Chapter 7.

2 BACKGROUND

This chapter addresses the main topics found in the literature related to our work. A complete review is out of the scope of this manuscript. Hence, we concentrate our attention on explaining the ideas upon which this work is based. Further literature is also found in subsequent chapters.

2.1 COLOR SPACES

A color space is a way of representing colors. The choice of color space may influence the compression performance by removing redundancies between colors [13]. As we employ color space conversion in this work, we briefly summarize the main color spaces in use today:

- **RGB:** RGB originated from cathode ray tube display applications, and it is most commonly used for storing and representing digital images since the human eye naturally captures the three primary colors used for image representation (red, green and blue), which gave the name to this color space [14]. The RGB representation was standardized in 1930 by the *Commission Internationale de l'Eclairage* (CIE), using the primary colors of red (700.0 nm), green (546.1 nm) and blue (435.8 nm) [13].

- **Perceptual Color Spaces:** Developed in the 1970s for computer graphics applications, hue-saturation color spaces were introduced to numerically represent the artistic idea of tint, saturation and tone [15]. The dominant color (for example, red, yellow, and purple) is represented by the Hue (or Tint), while saturation represents how pure the color is, starting with gray when the saturation is zero and going up to pure color when the saturation is maximum. The third component (Intensity, Value, or Lightness) measures how bright the color is.

  They cannot be directly described by the RGB space, but many non-linear transformations were proposed to relate them with the RGB space, for example [16]:

$$H = \cos^{-1}\left(\frac{1}{2}\frac{(R-G)+(R-B)}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right), \tag{2.1}$$

$$S = 1 - 3\frac{\min(R,G,B)}{R+G+B}, \tag{2.2}$$

$$V = \frac{R+G+B}{3}. \tag{2.3}$$

- **Chrominance Based Spaces:** YCbCr is an orthogonal color space, commonly used for image compression, that uses statistical independent components aiming to reduce the redundancy of RGB [13]. Color is represented by *luma* (Y), which is the luminance computed

from a weighted sum of RGB values, and two chrominances: chrominance blue (Cb) and chrominance red (Cr) that are computed by subtracting *luma* from red and blue components. Luma and chrominances are computed as:

$$Y = 0.299R + 0.587G + 0.114B, \tag{2.4}$$
$$C_b = B - Y, \tag{2.5}$$
$$C_r = R - Y. \tag{2.6}$$

The explicit separation between luminance and chrominance and its simplicity makes YCbCr a popular choice. Other similar color spaces are YCgCr, YIQ, YUV, and YES, which differ from YCbCr in how the chrominance is calculated. YUV, for example, in the BT.601 standard-definition television [17], is computed as

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.7}$$

The linear transformation that converts RGB to a chrominance space and the use of statistically independent components makes it very popular for image compression.

2.2 POINT CLOUDS

A point cloud (PC) consists of a set of data points in space. With a significantly large number of points, it is possible to represent the hull of objects and scenes. See Figure 2.1 for instance. As the number of points in the set increases, the hull of the Roman oil lamp becomes clearer.

12861 points 25722 points 257211 points

Figure 2.1: Point cloud "RomanOilLight" (University of São Paulo point cloud dataset[1]) with different number of points in the set.

Every point has some attributes associated with it, depending on its purposes. Its geometrical position (X, Y, and Z coordinates) is always present. We refer to this attribute only as of the point cloud geometry. Other attributes such as color and it's normal vector are also commonly used.

[1] http://uspaulopc.di.ubi.pt/

If beyond these attributes, we also have the connectivity among points we can form a mesh, that requires fewer points to satisfactorily represent an object (see Figure 2.2). In a polygon mesh, a subset of three or more points is chosen as the vertices of a polygon. These polygons are then used to represent the hull of the objects. Usually, a point selected as vertice for a given polygon is also a vertice for one or more neighboring polygons.

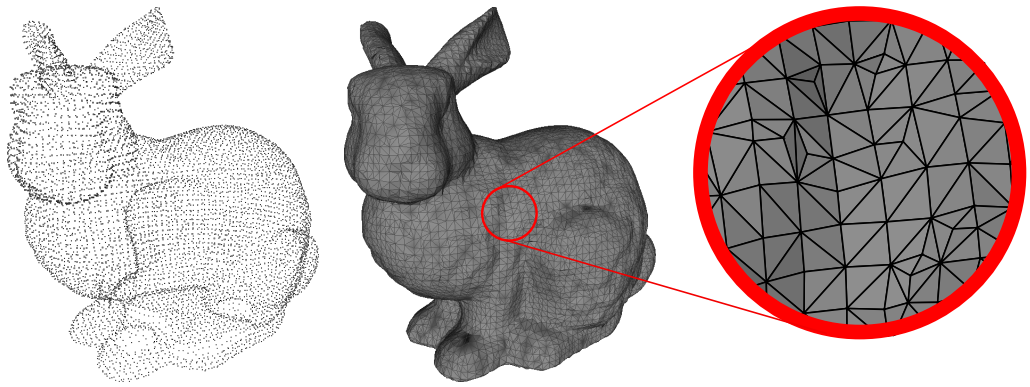


Figure 2.2: Stanford Bunny mesh.

The surface mesh can be derived from a point cloud using several techniques of surface reconstruction. Technologies in this regard range from methods that assume a well-sampled point cloud generalize to arbitrary shapes and produce a watertight surface mesh, to practices that make very loose assumptions on the quality of the point cloud, and operate on specific classes of shapes [18]. Polygonal meshes are easy to edit and ideal for synthetic 3D objects. The compression of such data can be done via a large number of existing methods [19]. However, the computational cost of the conversion process makes it difficult to be applied in real-time applications. Therefore, many researchers have opted to directly compress point clouds instead of polygon meshes.

Point clouds can be created from synthetic data, where a 3-dimensional model is used, but its more practical use resides in the capture of real-world scenes. They can be captured using an array of stereo cameras such as the ones used by 8i Labs[2] (see Figure 2.3) where the disparity between cameras gives us information about how far the objects are from the cameras and are used to construct the point cloud geometry.

The capture can be combined with Kinetic cameras, that projects an infrared light at the scene to perceive the distance of the objects to the camera without the need to estimate disparity [21]. The combination creates a RGB-D (RGB image plus per-pixel depth) camera that capture images in real time.

The array of cameras is used to capture point clouds in a small, usually indoor environment. On open environments, such as thoes of autonomous navigation, LiDAR is commonly used [22]. LiDAR is a term that originated from the combination o “light” and “radar”, although now most sources treat this word as an acronym for “Light Detection And Ranging” [22]. It is a process that collects measurements used to create 3D models and maps of objects and environments. It is

[2]https://www.8i.com/

Figure 2.3: Array of cameras capturing a point cloud at 8i [20].

widely used in many areas for applications in autonomous vehicle navigation [23], environment monitoring [22], topography, cultural heritage documentation, and others.

LiDAR is more of a methodology than a technology. It describes tools that employ ultraviolet, visible, or near-infrared light to sense the environment. It works very similar to a radar, where spatial relationships and shapes are deducted by the time it takes for light pulses emitted by a source to bounce off objects and return to the scanner. The direction and distance of whatever the pulse hits are recorded as a point of data. Data can vary from sparse to dense point clouds.

The points are a primitive concept upon which the geometry is built. The geometric points do not have any length, area, volume, or any other dimensional attribute. A common interpretation is that the concept of a point is meant to capture the notion of a unique location in Euclidean space. Therefore, for rendering, the points in the point cloud are rather represented by spheres. A more convenient way to represent this data is to employ *voxelized* point clouds (VPC). A *voxel* is the 3-dimensional (3D) extension of a pixel in an image. As the pixel is a square, the fundamental element of an image, a voxel is a cube and is also the fundamental element for what we could call as 3D images.

For simplicity, we assume that the object or the scene we want to represent is comprised of a cube of dimensions $W \times W \times W$, where W is a positive integer. If we divide this cube by W segments along each dimension we obtain W^3 smaller cubes of dimension $1 \times 1 \times 1$ that are referred, by definition, as voxels. Volumetric units of voxels can represent cubes of any real-

world size. Thus, it is common to employ a frame-to-world scale, keeping the voxels with their $1 \times 1 \times 1$ size, for convenience. The advantage of doing so is that now the positions of each voxel are discrete instead of continuous.

In contrast with a 2D image, in the 3D representation using voxels, most of them must be transparent. Otherwise, we would not be able to see inside the $W \times W \times W$ space we are considering. Also, many voxels in the interior of the objects we are representing may be impeded. They can be regarded as transparent, reducing the number of data required to portray an image. Typically, less than 10% of the voxels are not transparent. Those voxels that are transparent are commonly referred to as void voxels, or non-occupied, in an allusion that we do not have the color information for them. Those voxels that are not transparent are referred as occupied voxels (or simply voxels if we ignore the existence of void voxels).

The process that converts a standard point cloud to a voxelized point cloud is called voxelization. The space where the point cloud resides is divided in a regular grid representing the voxels position and points are assigned to voxels depending on their geometry, *i.e*, if a point fall within the region delimited by a voxel it is assigned to that voxel. Take, for instance, Figure 2.4. Five points are dispersed in the space, each one with a color attribute associated with it. The space around the points was divided into four voxels. Points are attributed to each voxel according to their position. Three cases may occur:

1. Just one point falls inside the region delimited by the voxel (upper-left voxel in Figure 2.4). In this case, the color attribute of that point is directly associated with the voxel;

2. More than one point fall inside the region delimited by the voxel (upper-right and bottom-left voxel in Figure 2.4). In this case, the average voxels' color is associated with the voxel;

3. No voxel falls inside the region delimited by the voxel (bottom-right voxel in Figure 2.4). In this case, the voxel does not have a color associated with it and is left transparent (it is also said to be a void voxel, in contrast to occupied voxels).

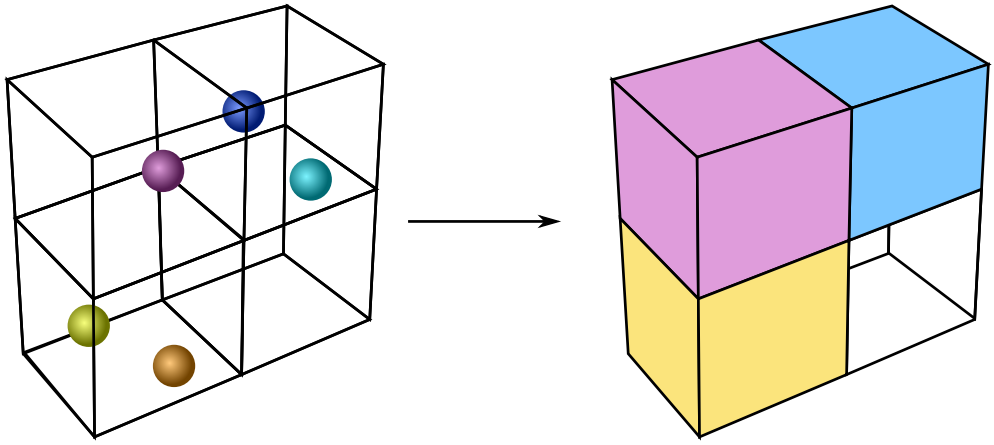


Figure 2.4: Illustration of the voxelization process.

Due to its sparsity, a voxelized point cloud can be conveniently represented in the form of a list of the occupied voxel attributes. In one of the simplest scenarios, the voxels' attributes are its euclidean position, also referred as geometry and its color in RGB, YUV, or any other color space. This is the way point clouds are stored in the "Polygon File Format", also known as the "Stanford Triangle Format", whose extension is PLY. In the PLY files, two main variants are available:

- The binary format, where the attributes are written as `char`, `short`, `int`, `float` or `double` and their respective unsigned variant, if available;
- The ASCII format, where the attributes are written as text. This format requires more bits, but is human readable.

Figure 2.5 shows an example of a PLY file. The first part of the file is a header, always written as text, indicating the format used, the number of points in the point cloud, attributes of the point cloud, and comments. The header ends at the line containing the `end_header`. The list of attributes follows the header. In this particular case is used the ASCII format, and each line corresponds to one point (or voxel). The first three values are the x, y, and z coordinates of the point, and the last three values are the red, green, and blue values of the color attribute. The binary format is similar, but, after the end header is given, there is a sequence of bits representing those values without any line break in between.

Point clouds may need a massive amount of memory to be stored as a PLY file format. A typical point cloud we consider in this work has around $500,000$ and $4,000,000$ points. In the binary format, the PLY file uses 24 bits to store the color attribute (three unsigned 8 bits integer for color component) and 96 bits to store the geometry attribute of each point (three 32 bits single floating point value). In the ASCII format, more bits are required.

Historically, the compression of point clouds (*voxelized* point clouds) has been divided into two major tasks: the compression of the geometry attributes and the compression of the color attributes. These two attributes have been independently encoded to compressed files.

2.3 GEOMETRY ENCODER

Examples of geometry encoders are the wavelet-transform-based scheme of Ochatta and Saupe [24], and the octree-based geometry compression algorithms [25, 26]. The latter is the best-known method. The geometry attribute of the point clouds can be encoded using the octree scanning of the voxels, which has been shown to be very efficient [27].

The octree is a geometric 3D modeling technique [28] that is the 3D extension of a 2D quad-tree. Our encoding data is confined in a cube with dimensions $W \times W \times W$ wherein ours voxels lay. From Figure 2.6-(a), we start with a list of voxels inside such a region, which is then divided in half along one of its dimensions (let us assume x axis, for example), dividing the original cube

```
ply
format ascii 1.0
comment Version 2, Copyright 2017, 8i Labs, Inc.
comment frame_to_world_scale 0.179523
comment frame_to_world_translation -45.2095 7.18301 -54.3561
comment width 1023
element vertex 792192
property float x
property float y
property float z
property uchar red
property uchar green
property uchar blue
end_header
235 55 63 126 111 96
234 57 63 126 109 92
235 56 63 126 111 95
235 57 63 125 108 92
235 58 63 133 115 99
236 56 63 123 109 97
236 57 63 121 106 94
237 56 63 119 105 93
237 57 63 121 107 94
236 58 63 124 107 92
236 59 63 123 106 92
237 58 63 122 106 94
237 59 63 121 104 91
```

Figure 2.5: Example of a PLY file. Only the 27 first lines of the file are shown. This file in particular represents 792192 point and is represented by its geometric and color attribute only.

precisely in half (Figure 2.6-(a)). At this stage, we say that we moved up one level on this binary tree. The list of occupied voxels is divided into two lists: one for the voxels laying in the leftmost half and another list for the rightmost half. We continue our division along another dimension (y axis, for example), dividing into four regions and producing four voxel's lists. Finally, repeating the process for the remaining dimension (z axis, for example), we obtain eight cubes similar to the original one, but with a width of $W/2$. At this stage, we say that we move one depth-step in this binary tree (1 depth-step = 3 levels).

We can continue with this process, further dividing the space. At depth d, we have cubes of size $W/2^d$. If W is chosen to be a power of 2, at the depth d where $2^d = W$, the resulting cubes have size $1 \times 1 \times 1$ and accommodate one single voxel, at maximum. For this reason, most voxelized point clouds have W chosen to be a power of 2. The higher the value of W, the

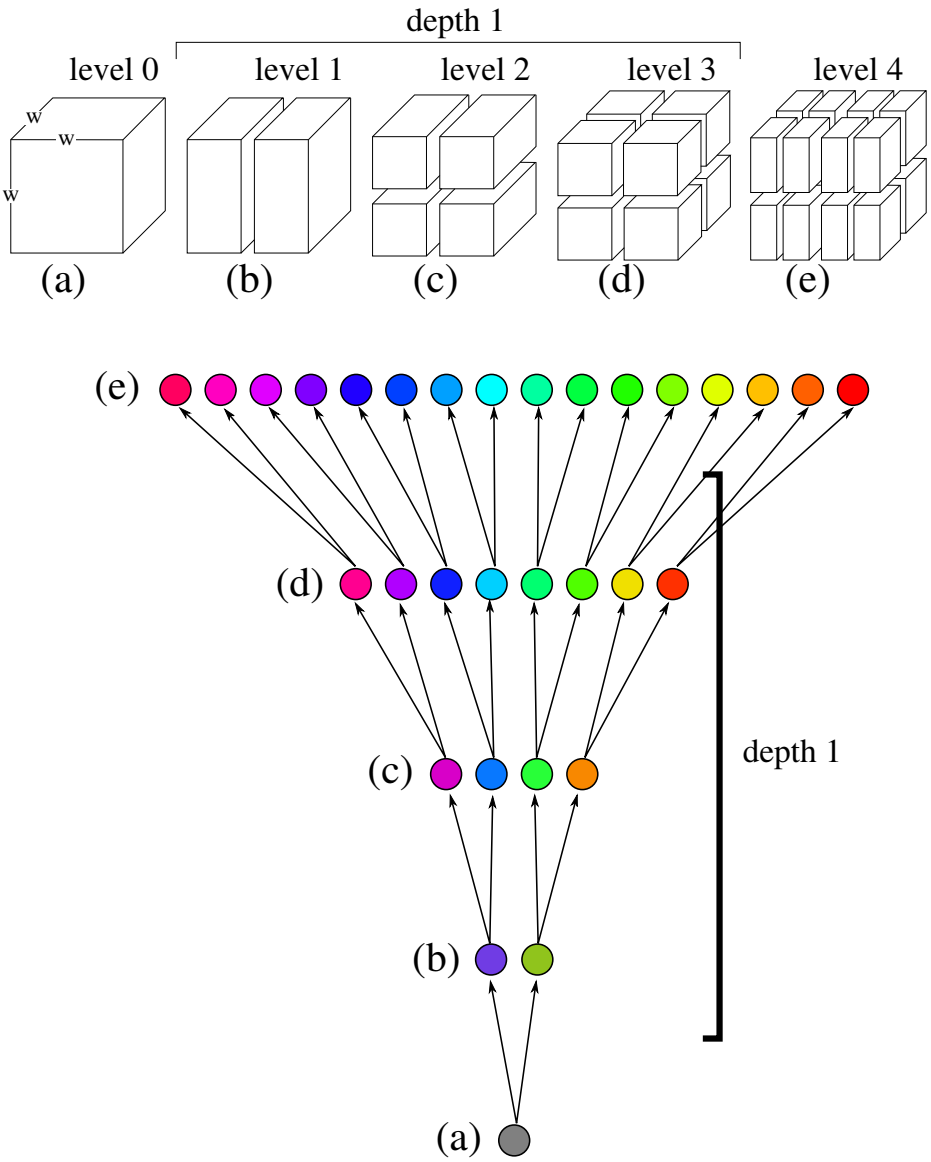


Figure 2.6: Octree scanning of voxels. In (a), we have a space of size $W \times W \times W$ that is divided in half, forming two sub-regions. Equivalently, in the binary tree, we start at the tree root at (a) that is divided into two nodes in (b). From (b) to (c), the regions are subdivided further along another dimension. Each node becomes two nodes after every division. Finally, from (c) to (d), the sub-regions are cubic with dimensions $W/2 \times W/2 \times W/2$, and, at this point, we say that we have moved one depth-step up the tree. We can continue dividing the space even further, until reaching the desired resolution.

more detailed the point cloud is, since it comports more voxels, hence more bits are necessary to encode it.

These sub-lists of voxel attributes may be empty for some of the divided regions if there are no voxels inside it. Therefore, there is no need to sub-divide this region any further as it makes no difference. We can signal if a given area is occupied by any voxels or not by a bit that is `1` in the case of an occupied region and `0` otherwise. Furthermore, we can ignore the sub-division along with each level and only take into account the division of when we go to one depth to another. When traveling one depth in the tree, any cubic region is divided into eight sub-cubes with half

its width.

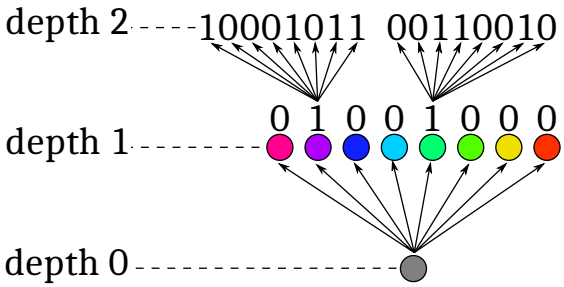


Figure 2.7: Octree signaling. Once a node is divide in 8 nodes, one byte is used to signalize which nodes are occupied. At $depth = 1$, only two nodes are occupied: the second and the fifth, signalized by the byte `01001000`. Only the occupied nodes are further divided and we send one byte for each occupied node at $depth = 1$. In this example, assuming $depth = 2$ is the last division of this geometry, the sequence `0100100010001011100110010` is enough to encode the position of the 7 voxels in this tiny point.

In Figure 2.7, the original list of voxel is divided into eight lists after going from depth 0 to depth 1. Some of those lists are empty, and it is indicated by a bit. Therefore, a total of 8 bits, or one byte, is used. When going from depth 1 to depth 2, we need to take into account the non-empty lists, that are further divided into eight nodes and we attribute one byte to each. When attaining depth d, where $2^d = W$, every sub-division corresponds to precisely one voxel. By knowing which voxels are occupied and which are void, we can reconstruct the geometric information of the voxelized point cloud. The sequence `0100100010001011100110010` is enough to encode the position of the seven voxels in that tiny point cloud example given in Figure 2.7.

This method to encode the voxelized point cloud geometry can be combined with other techniques, such as dictionary entropy coders (*e.g.* Lempel–Ziv–Welch) to reduce the redundancy among bytes. The octree demands between 2 and 3 bits per occupied voxel to encode the geometry [29].

The method explained here is a lossless geometry compression. Compression is achieved by eliminating the need to signal individually which voxels in the point cloud are occupied. This also allows a progressive representation of point clouds since the user can stop at any given level [28,30]. The coder can also further compress the data by exploiting the inherent redundancy by using intra-frame [31] or inter-frame context [30]. In video coding, intra-frame redundancy refers to the ability to predict portions of an image based on previously encoded parts of this same image. Inter-frame redundancy, on the other hand, exploits the information of an already encoded frame of the same video to predict the information of the frame being currently encoded.

Lossy compression methods are also under development. The main problem resides in how to measure the quality of the degraded geometry. Two geometric distortion metrics are widely used: point-to-point and point-to-plane [32]. In the former, a nearest neighbor search is used to find the correspondence between points in the original point cloud with points in the reconstructed one. Referring to Figure 2.8, the point A_i from the original point cloud correspond to the point B_j in the reconstructed point cloud. The error vector is $\overrightarrow{\epsilon_i}$. The mean squared error ($MSE_{ori \to rec}$) is

computed from the norm of $\overrightarrow{\epsilon_i}$. We also compute the mean squared error from the correspondence between the reconstructed point cloud and the original ($MSE_{rec \to ori}$). Finally, the point-to-point metric is computed and can be presented in two forms:

1. **In terms of the mean squared error:** it is computed from the maximum value between $MSE_{ori \to rec}$ and $MSE_{rec \to ori}$;
2. **In terms of peak signal to noise ratio:** it is computed as

$$10\ log_{10}\left(\frac{3D^2}{max(MSE_{ori \to rec}, MSE_{rec \to ori})}\right), \tag{2.8}$$

where D is the highest distance between points in the original point cloud.

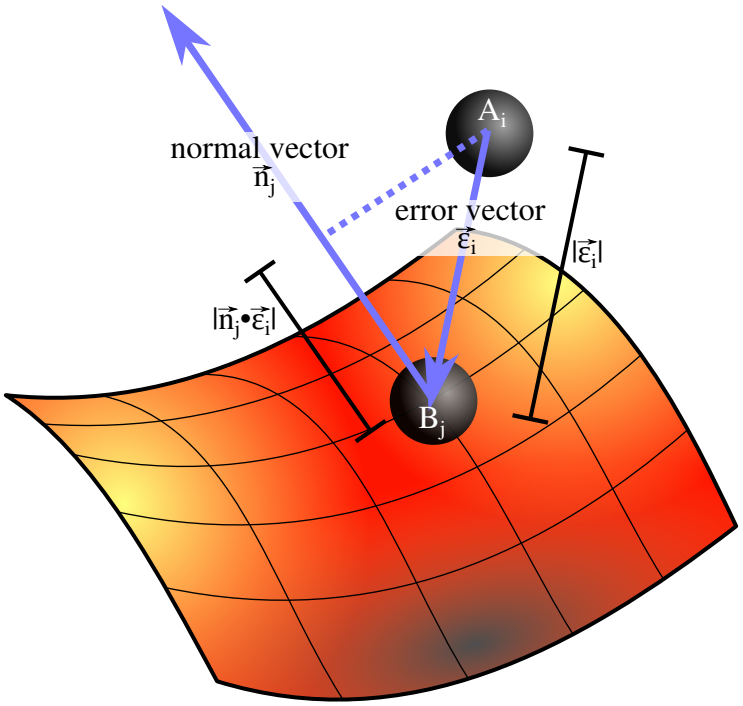


Figure 2.8: Point-to-point and point-to-plane distortion measure. The points of one point cloud are corresponded to the points of a reference point cloud. In the point-to-point metric is used the norm of the error vector to compute the MSE, while in the point-to-plane metric is used the projection of the error vector along the surface normal vector of the reference point.

The point-to-plane metric differs from the point-to-point in how the error is computed. The error vector is projected along the surface normal vector of the corresponding point. In this way, the metric ignores small errors in the point position as long as it stays within the surface of the object being represented by the point cloud. The point-to-plane metric can also be expressed in terms of the mean squared error or peak signal to noise ratio.

To compute the point-to-plane metric, each point must have a normal. The normals of the original point cloud, if not present, can be estimated by finding the parameter of a plane that best fits the neighborhood of each point. The reconstructed point cloud can have its normals assessed

by either fitting a plane to each point or from transferring the normal of the corresponding point in the original point cloud.

Both the point-to-point and the point-to-plane metrics are currently in use by MPEG in the efforts of point cloud compression standardization [33]. According to Alexiou *et. al* [34], the point-to-plane metric expressed in terms of MSE is the one that best correlates with subjective quality.

2.4 COLOR ENCODER

The color can be encoded separately from the geometry as long as we preserve an agreed ordering between encoder and decoder for them to correctly associate the color to its corresponding geometry (which is usually encoded by the octree-based geometry compression).

Color compression is a challenging problem since these attribute signals are unstructured. One of the most successful early attempts in color compression was the graph transform [35]. It was later replaced by the Region-Adaptive Hierarchical Transform [29] (RAHT), a coder with similar performance and a smaller complexity, which was then improved, having a performance above that of graph transform maintaining its low complexity [36].

2.4.1 Graph transform

In the graph transform, we construct graphs on small neighborhoods of the point cloud by connecting nearby points. The graph is a generic structure where a set of objects are, in some sense, connected. The objects correspond to mathematical abstractions called vertices, nodes, or points, and each of the related pairs of vertices is called an edge [37]. A weight is attributed to an edge that measures how strong those nodes are connected.

The voxelized point cloud is divided into blocks, and for each block is constructed a graph by connecting all voxels that are with a distance of 1 along any axis, as shown in Figure 2.9, a 2D example for simplicity. Along each edge is attributed a weight that is equal to the inverse of the distance. In a 3D space, the allowed weights are 1, $1/\sqrt{2}$ and $1/\sqrt{3}$.

A precision matrix $\mathbf{Q}$ is constructed for the graph. The element q_{ij} in the i-th row and j-column of $\mathbf{Q}$ gives the relationship between the i-th and j-voxel and is expressed as:

$$q_{ij} = \delta \begin{cases} \sum_n w_{in} & \text{if } i = j \\ -w_{ij} & \text{otherwise} \end{cases}, \tag{2.9}$$

where δ is a scalar that is related to the graph signal's variance and w_{ij} is the weight of the edge connecting the i-th voxel to the j-th voxel. Note that $w_{ij} = w_{ji}$.

In the example given in Figure 2.9, the precision matrix is

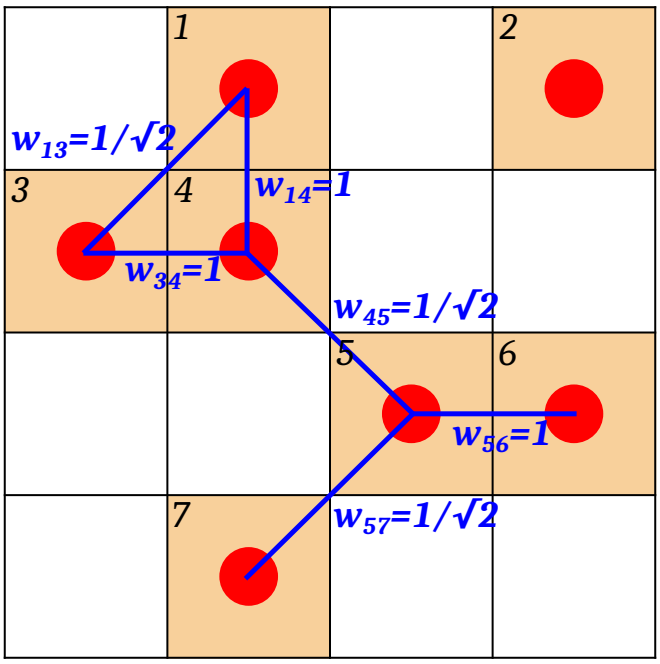


Figure 2.9: Construction of the graph upon a 4×4 2D voxelized space.

$$\mathbf{Q} = \delta \begin{bmatrix} w_{13}+w_{14} & 0 & -w_{13} & -w_{14} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -w_{13} & 0 & w_{13}+w_{34} & -w_{34} & 0 & 0 & 0 \\ -w_{14} & 0 & -w_{34} & w_{14}+w_{34}+w_{45} & -w_{45} & 0 & 0 \\ 0 & 0 & 0 & -w_{45} & w_{45}+w_{56}+w_{57} & -w_{56} & -w_{57} \\ 0 & 0 & 0 & 0 & -w_{56} & w_{56} & 0 \\ 0 & 0 & 0 & 0 & -w_{57} & 0 & w_{57} \end{bmatrix}. \tag{2.10}$$

Remark that the precision matrix of the example has a zero determinant because voxel 2 has no connections. This is not a problem for the transform.

The precision matrix is the inverse of the covariance matrix in a typical multivariate Gaussian distribution. It is used to define a Gaussian Markov Random Field. Suppose we have a vector $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$ describing the voxels' color. The probability of $\mathbf{x}$ is given by

$$P(\mathbf{x}) = \frac{|Q|^{\frac{1}{2}}}{(2\pi)^{\frac{N}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \mathbf{Q}(\mathbf{x}-\mu)}, \tag{2.11}$$

where μ is the average of $\mathbf{x}$.

For a signal following the Gaussian Markov Random Field model with the precision matrix $\mathbf{Q}$, the best transform to separate (decorrelate) this signal is the Karhunen-Loève Transform (KLT), obtained as the eigenvector matrix Φ of the precision matrix $\mathbf{Q}$ decomposition.

The original $W \times W \times W$ space is divided into blocks. Each block is transformed using the

eigenvector matrix Φ, resulting in a vector $\mathbf{y} = \Phi\mathbf{x}$ of coefficients for each of the three color components (YUV), that are separately encoded.

The first few coefficients of $\mathbf{y}$ in each block correspond to DC terms. For instance, if the block has m disconnected subsets of points, the first m coefficients in $\mathbf{y}$ correspond to DC coefficients. The remaining coefficients in $\mathbf{y}$ are AC terms.

These vectors are quantized as $\mathbf{y_Q} = round(\mathbf{y}/Q_{step})$, where Q_{step} is the quantization step size. The AC coefficients of $\mathbf{y_Q}$ are then entropy coded with the use of a simple arithmetic encoder [38,39] assuming an underlying zero mean Laplacian probability distribution. The variance of the Laplacian distribution is estimated from the inverse of the eigenvalues related to each coefficient.

The DC coefficients of $\mathbf{y_Q}$ are also arithmetic coded assuming a Laplacian distribution, but first removing their average value. The variance of the distribution is adaptively updated as the coefficients are encoded.

Queiroz and Chou [40, 41] proposed a very similar approach but using a gaussian process to determine the covariance matrix upon which the KLT transform is derived. They propose four models:

1. **Non-Parametric**

   The covariance function $R(\mathbf{d})$, where $\mathbf{d}$ is the displacement between voxels, is used to describe this model. $R(\mathbf{d})$ is assumed to be symmetrical along every axis. To enforce this, instead of estimating $R(\mathbf{d})$, it is estimated the $R(d) = R(|\mathbf{d}|)$ from an unbiased estimator.

2. **Ornstein-Uhlenbeck (OU)**

   The OU model depends on a single parameter $0 < \rho < 1$, which describes how the covariance decays with distance. The covariance function is modeled by $R(\mathbf{d}) \propto \rho^{|\mathbf{d}|}$. The parameter ρ is estimated from the voxelized point cloud by maximum likelihood. Authors have observed that a value of $\rho = 0.95$ has led to better compression performance for most point clouds they tested for their encoder [41]. The constant value that converts the proportionality to equality is unimportant as this parameter is not necessary to compute the KLT.

3. **Inverse Distance**

   The inverse-distance model estimates the precision function $Q(\mathbf{d})$ instead of the covariance function directly. Note that the precision and covariance functions are inverse of each other [41]. The precision function is computed as given in Equation (2.9).

4. **Auto-Regressive**

   This model considers that the color of a voxel depends linearly on the color of its neighbors and on a stochastic term. The linear combination is a prediction of the voxel color, while the stochastic term introduces the intrinsic imperfections. The model is a conditional

auto-regression whose parameters are the variance of the stochastic term, assumed to be a Gaussian white noise, and the terms of the linear combination. These parameters are used to compute the precision matrix.

For the correct decodification, all models require sending models parameters to the decoder as side information. The inverse distance model results in a compression method that is essentially the same as the graph transform [35]. In their experiments, it was observed that the Non-Parametric and Ornstein-Uhlenbeck models outperform both the Inverse Distance and Auto-Regressive models in terms of energy compaction, transform coding gain, and distortion-rate performance.

2.4.2 Region-Adaptive Hierarchical Transform

One of the purest color encoder is the Region-Adaptive Hierarchical Transform. It is a hierarchical sub-band transform that resembles a variation of the 3D discrete Haar wavelet transform [42]. Therefore, to understand the RAHT, let us first explain the 1D Haar transform that is later generalized for multidimensional signals. A straightforward way to do so is through an example. Let us take the signal

$$\mathbf{x} = (x_0, x_1, \ldots, x_{N-1}), \tag{2.12}$$

containing N elements.

The wavelet transform decomposes the signal into two others:

$$\mathbf{h} = (h_0, h_1, \ldots, h_{N/2-1}), \tag{2.13}$$

a low pass version of $\mathbf{x}$, and

$$\mathbf{g} = (g_0, g_1, \ldots, g_{N/2-1}), \tag{2.14}$$

a high pass version of $\mathbf{x}$. Each of these signals having $N/2$ elements (N assumed to be even). The coefficients of $\mathbf{h}$ and $\mathbf{g}$ are obtained by a linear transformation given by:

$$\begin{bmatrix} h_n \\ g_n \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_{2n} \\ x_{2n+1} \end{bmatrix}, \tag{2.15}$$

for the n-th coefficient. In this sense, h_n is the average of x_{2n} and x_{2n+1} (with its amplitude amplified by $\sqrt{2}$) and g_n is their difference. The inverse transform is given by:

$$\begin{bmatrix} x_{2n} \\ x_{2n+1} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} h_n \\ g_n \end{bmatrix}. \tag{2.16}$$

The signals $\mathbf{h}$ and $\mathbf{g}$ are usually concatenated together as $(\mathbf{h}|\mathbf{g})$, resulting in a signal with N coefficients, the same as the number of elements in $\mathbf{x}$. The process can be applied again over $\mathbf{h}$, resulting in a low and high pass signal with $N/4$ coefficients each. We can keep repeating this process over the low pass signal, at each step dividing the number of low pass coefficients by 2.

The Haar is an orthonormal transform because its base vectors, $(1,1)/\sqrt{2}$ and $(-1,1)/\sqrt{2}$, are unitary vectors orthogonal to each other. A multidimensional Haar transform is obtained by applying the transform independently along each dimension. For a 2D image, for example, the Haar transform is applied first along the columns of the image resulting in the image shown in Figure 2.10-(b), with the low pass signal in the up side. Then, we apply the Haar transform in the resulting concatenated image along the lines, resulting in the image shown in Figure 2.10-(c). After this first step, we can continue applying the Haar transform in the resulting low-pass image, as depicted in Figure 2.10-(d).

The RAHT applies similar steps over the 3D voxel's space as the Haar transform but coping with the fact that a voxelized point cloud is a very sparse signal where most of the voxels are void. The sparsity is taken into account using a parameter called weight. Initially, the weight is set to be 1 to all occupied voxels and 0 for void voxels. The voxels are then combined two by two, along with each dimension, mimicking the Haar transform process. Let us denote by $F^{\ell}_{i,j,k}$ the color of the voxel at the position $x=i$, $y=j$ and $z=k$ at level ℓ and by $w^{\ell}_{i,j,k}$ its corresponding weight. Along the x-dimension, the RAHT transform is given by:

$$\begin{bmatrix} F^{\ell}_{i,j,k} \\ G^{\ell}_{i,j,k} \end{bmatrix} = \begin{bmatrix} a & b \\ -b & a \end{bmatrix} \begin{bmatrix} F^{\ell+1}_{2i,j,k} \\ F^{\ell+1}_{2i+1,j,k} \end{bmatrix}, \tag{2.17}$$

where

$$a^2 = \frac{w^{\ell+1}_{2i,j,k}}{w^{\ell+1}_{2i,j,k} + w^{\ell+1}_{2i+1,j,k}}, \text{ and } b^2 = \frac{w^{\ell+1}_{2i+1,j,k}}{w^{\ell+1}_{2i,j,k} + w^{\ell+1}_{2i+1,j,k}}. \tag{2.18}$$

Of course this equation only make sense when at least one of the voxels in the pair is not void, otherwise a division by zero would occur since $w^{\ell+1}_{2i,j,k} = w^{\ell+1}_{2i+1,j,k} = 0$ in this case. The combination of two void voxel are of no interest as they do not carry any information.

The inverse transform is then given by

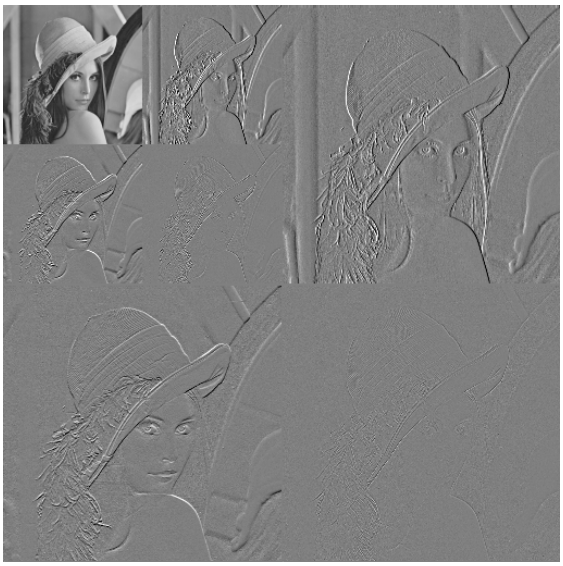
$$\begin{bmatrix} F^{\ell+1}_{2i,j,k} \\ F^{\ell+1}_{2i+1,j,k} \end{bmatrix} = \begin{bmatrix} a & -b \\ b & a \end{bmatrix} \begin{bmatrix} F^{\ell}_{i,j,k} \\ G^{\ell}_{i,j,k} \end{bmatrix}. \tag{2.19}$$

The transform matrix given in Equation (2.17) reduces to that given in Equation (2.15) when $w^{l}_{2i,j,k} = w^{l}_{2i+1,j,k}$. If $w^{l}_{2i,j,k}$ or $w^{l}_{2i+1,j,k}$ is equal to zero, the low pass coefficient, $F^{l}_{i,j,k}$ will simply copy the color of the non-void voxel, while the high-pass coefficient will copy the color of the void voxel. As a void voxel has no color information, this process ends up creating no high pass coefficient in this case, and the transform does not need to be computed. In other words, when

(a) Original image

(b) Transformed coefficients after applying Haar transform over the vertical axis

(c) Transformed coefficients after applying Haar transform over the columns and then horizontal axis



(d) Similar result after applying a second separable Haar transform over the low-pass image

Figure 2.10: 2-dimensional discrete Haar transform applied on Lena.

one of the voxels in the pair is void, the attribute of the other is passed to the next level.

To the low pass coefficient promoted to the level above, $F^{\ell}_{i,j,k}$, is associated with the weight

$$w^{\ell}_{i,j,k} = w^{\ell+1}_{2i,j,k} + w^{\ell+1}_{2i+1,j,k}, \tag{2.20}$$

when processing along the x-axis.

Because of the bypass of the void voxels, RAHT is non-separable. The high-pass coefficient is not subject to further processing along the other dimension. They are sent directly to the next

compression steps, which are quantization and entropy coding. The low-pass coefficients, on the other hand, are treated as new voxels. The process is repeated along each dimension (x, y and z), advancing one level at a time, until reaching the level where only one low-pass coefficient remains: the DC coefficient. The DC coefficient is sent to the decoder, along with all the high-pass coefficients generated in the process.

To better illustrate this transform, take the example in Figure 2.11 of a 2D (for simplification) voxelized point cloud. The color is assumed to be monochromatic, but the process is analogous for each color component.

$w^4_{0,3} = 0$	$F^4_{1,3} = 10.00$ $w^4_{1,3} = 1$	$F^4_{2,3} = 9.00$ $w^4_{2,3} = 1$	$F^4_{3,3} = 7.00$ $w^4_{3,3} = 1$
$F^4_{0,2} = 12.00$ $w^4_{0,2} = 1$	$F^4_{1,2} = 14.00$ $w^4_{1,2} = 1$	$w^4_{2,2} = 0$	$w^4_{3,2} = 0$
$F^4_{0,1} = 10.00$ $w^4_{0,1} = 1$	$F^4_{1,1} = 11.00$ $w^4_{1,1} = 1$	$w^4_{2,1} = 0$	$F^4_{3,1} = 4.00$ $w^4_{3,1} = 1$
$F^4_{0,0} = 8.00$ $w^4_{0,0} = 1$	$w^4_{1,0} = 0$	$w^4_{2,0} = 0$	$w^4_{3,0} = 0$

Figure 2.11: Original 2D point cloud example (level $\ell = 4$), depicting the weight and color of each voxel in a 4×4 space. Cells representing occupied voxels are colored.

We apply the first step of the transform to the original point cloud, going from level $\ell = 4$ to level $\ell = 3$, by grouping voxels in pairs along the x-axis. This process generates two signals: a low-pass and a high-pass with the x dimension reduced by half that is depicted in Figure 2.12. Three cases may occur:

- Both voxels in the voxel pair being transformed are occupied. In this case, we generate both a low- and high-pass coefficient. Example: pair $F^4_{0,2}$ and $F^4_{1,2}$;
- Just one of the voxels in the voxel pair being transformed is occupied. In this case, the low-pass coefficient is a copy of the color of the occupied voxel, and no high-pass coefficient is generated. Example: pair $F^4_{0,3}$ and $F^4_{1,3}$;
- Both voxels in the voxel pair being transformed are void. I this case, neither a low- nor a high-pass coefficient is generated. Example: pair $F^4_{2,2}$ and $F^4_{3,2}$.

The weights of the voxels in the pair are combined, and we obtain a new point cloud in the low-pass signal. We then apply a similar process along with the y-axis, resulting in the low- and high-pass signals given in Figure 2.13. At this point, we have moved two levels over the tree. As this example is two dimensional, we have also moved one depth, obtaining a point cloud where both dimensions have been halved.

We repeat the process, resulting in the signal depicted in Figures 2.14 and 2.15.

low-pass		high-pass	
$F^3_{0,3} = 10.00$ $w^3_{0,3} = 1$	$F^3_{1,3} = 11.31$ $w^3_{1,3} = 2$		$G^3_{1,3} = -1.41$
$F^3_{0,2} = 18.38$ $w^3_{0,2} = 2$	$w^3_{1,2} = 0$	$G^3_{0,2} = 1.41$	
$F^3_{0,1} = 14.85$ $w^3_{0,1} = 2$	$F^3_{1,1} = 4.00$ $w^3_{1,1} = 1$	$G^3_{0,1} = 0.71$	
$F^3_{0,0} = 8.00$ $w^3_{0,0} = 1$	$w^3_{1,0} = 0$		

Figure 2.12: Resulting signals after applying the first step of the RAHT along the x-axis, going from level $\ell = 4$ to level $\ell = 3$. A low-pass and a high-pass signals are generated. The high-pass coefficients are ready to be processed to be sent to the decoder.

$F^2_{0,1} = 20.78$ $w^2_{0,1} = 3$ $F^2_{0,0} = 16.74$ $w^2_{0,0} = 3$	$F^2_{1,1} = 11.31$ $w^2_{1,1} = 2$ $F^2_{1,0} = 4.00$ $w^2_{1,0} = 1$	low-pass
$G^2_{0,1} = 2.45$		high-pass
$G^2_{0,0} = -2.04$		

Figure 2.13: Resulting signals after applying the second step of the RAHT along the y-axis on the low-pass signal generated at level $\ell = 3$.

low-pass	high-pass
$F^1_{0,1} = 23.25$ $w^1_{0,1} = 5$ $F^1_{0,0} = 16.49$ $w^1_{0,0} = 4$	$G^1_{0,1} = -4.38$
	$G^1_{0,0} = -4.91$

Figure 2.14: Resulting signals after applying the third step of the RAHT along the x-axis on the low-pass signal generated at level $\ell = 2$.

$F^0_{0,0} = 28.32$ $w^0_{0,0} = 9$	low-pass
$G^0_{0,0} = -3.21$	high-pass

Figure 2.15: Resulting signals after applying the fourth step of the RAHT along the y-axis on the low-pass signal generated at level $\ell = 1$. At this level we have only one low-pass coefficient, the DC value.

The DC and AC coefficients are then rastered onto a vector (the rastering of cofficients onto a vector is explained in chapter 3), quantized with a quantization step Q_{step} and entropy coded using an Arithmetic Coder. The pattern used for rastering does not influence the number of bits after entropy coding since they do not employ an adaptive coder. They assume that the coefficients obey a Laplacian distribution with zero average. Coefficients are separated in "buckets". Each bucket contains all coefficients with the same weight. The variance is estimated for each bucket, sent to the decoder as side information, and the coefficients are coded using the estimated variance.

2.4.3 Point cloud compression based on image and video codecs

State-of-art video compression algorithms can be used to compress point clouds if we are able to convert the PC to a 2-dimensional video and back, compressing geometry and color at the same time [43]. The advantage of this technique is the availability of dedicated hardware for video compression [44].

The technique is based upon the idea that the points in the point cloud represent the hull of an object. Therefore, these points can be divided in patches. The patches are generated by a clustering algorithm. First, the normal vector for each point is computed as described by Hoppe *et. al* [45]. Then, points are associated with one of the six faces of the $W \times W \times W$ cube, *i.e*, each each point is associated with the face of the cube that has the closest normal to the point. This initial clustering is iteratively refined based on the neighborhood of each point and, finally, patches are extracted by finding connected regions.

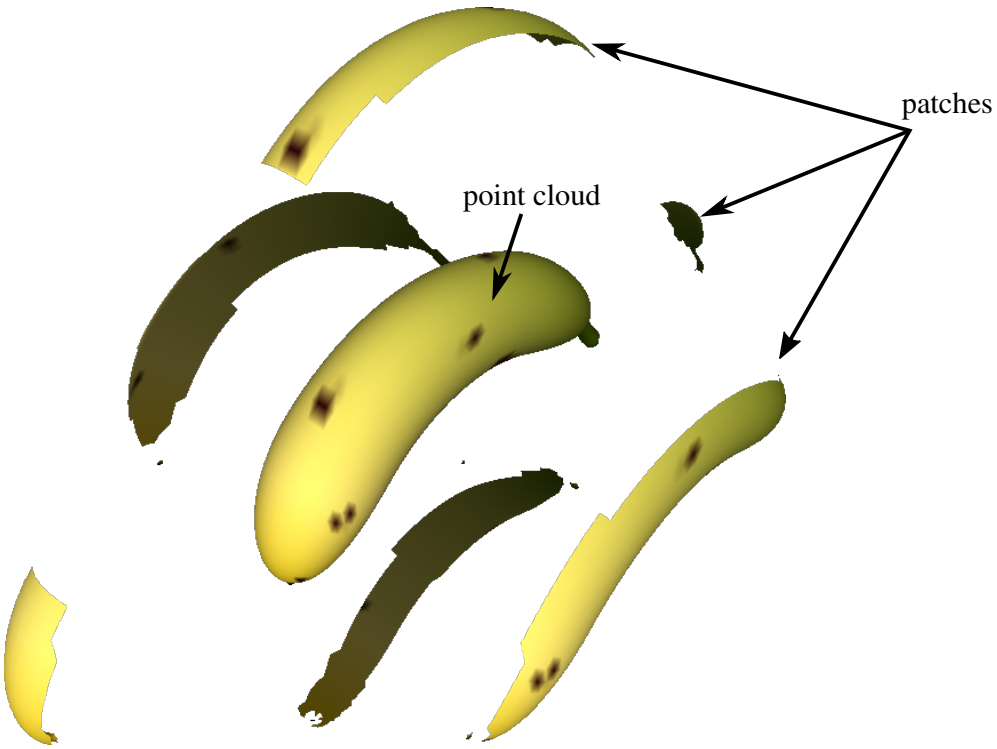


Figure 2.16: Patches generation from a point cloud.

From the patches, two images are generated, the texture image and the geometry image, by packing this patches onto a 2-dimensional grid. The patch location is determined through an exhaustive search that is performed in raster scan order. Figure 2.17 shows one example of texture image from the point cloud of a banana. The geometry image is monochrome and provides the depth of the points it represents.

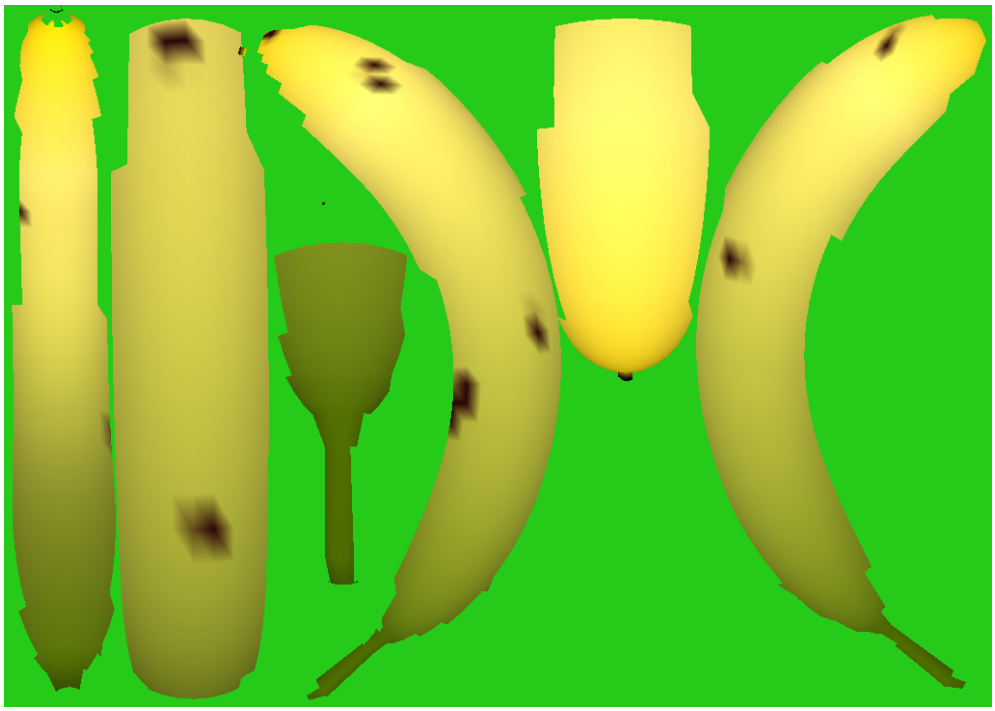


Figure 2.17: Example of texture image. The texture map was created from the patches in Figure 2.16.

The texture and geometry images plus some metadata required for the decoder to correctly interpret these images are then encoded using the High-Efficiency Video Coding (HEVC), a state-of-art video compression algorithm.

This is lossy a compression algorithm where both geometry and color information are degraded.

2.5 PLENOPTIC FUNCTION

In the field of light theory, we refer to image formation in terms of light rays [46]. Mathematically speaking, the complete light information of an environment can be expressed by a plenoptic function. The plenoptic function is a 7-dimensional function idealized in computer vision and computer graphics to represent how the image of a scene is viewed from any possible viewing position. It describes the intensity of light observed from every position and every direction in a 3-dimensional space [47].

In Figure 2.18 we represent a light ray emanating from point (x, y, z). The radiance along

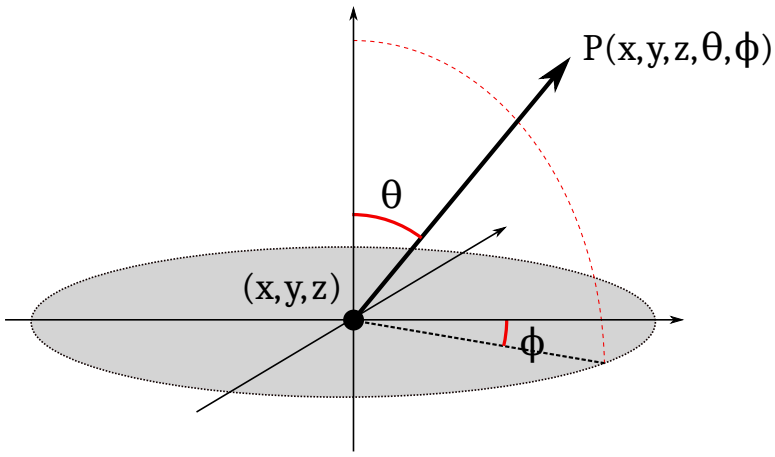


Figure 2.18: Plenoptic function.

such ray is given by:

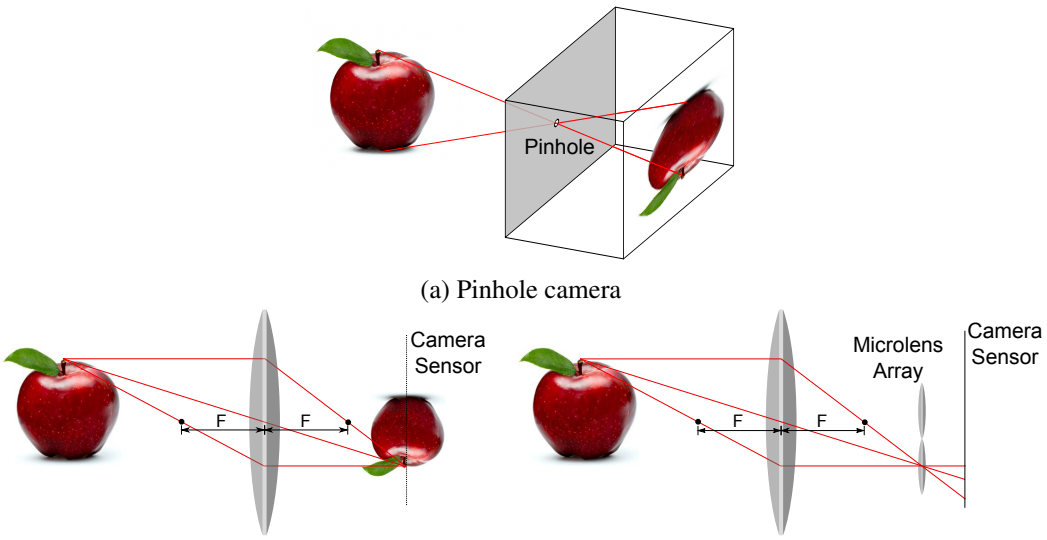
$$P(x, y, z, \theta, \phi), \tag{2.21}$$

where ϕ is the azimuth and θ the elevation angle. For a fixed point (x, y, z), we get the set of rays passing through a given point, which is referred to as a pencil of light.

Any camera can capture a sample of the light field, represented by the plenoptic function, but no single device is capable of capturing all dimensions of this function. A pinhole camera, for example, is the most straightforward device used to capture images, depicted in Figure 2.19-(a). It consists of a closed dark chamber with a tiny hole in one of its faces. The light emanating from the exterior goes through the hole and projects itself in the opposite front of the chamber. In this fashion, the pinhole camera is capturing only light rays going through the pinhole position. It is equivalent to capturing the $P(x, y, z, \theta, \phi)$ for a fixed (x, y, z) position, the position of the pinhole. That means that the plenoptic information sample captured by a pinhole camera has only 2 degrees of freedom, given by (θ, ϕ).

Conventional cameras (see Figure 2.19-(b)) use a lens instead of a pinhole, capturing a continuum of positions. The lens curvature bends the light rays that are imaged at the sensor plane. This results in more light entering the device when compared to a pinhole camera, thus reducing the noise effect. The sensor works as an integration device, combining all light rays falling on the same sensor pixel and losing any directional information, *i.e.* the angle at each of the rays are arriving at the sensor. In this sense, a traditional camera, using lenses, captures light information in a very similar way to that of a pinhole camera.

Another form of capturing the light field arises from a modification of a conventional camera as proposed by Gabriel Lippmann in 1908 [48], shown in Figure 2.19-(c). An array of microlenses is placed before the sensor, deviating the rays hitting each microlens as a function of its incident angle. As a consequence, the position where the light arrives at the sensor plane gives information about the spatial location of the ray (which microlens it crossed) and its incidence angle. This



(a) Pinhole camera

(b) Traditional camera

(c) Light-field camera

Figure 2.19: Imaging of a scene by a traditional lens camera and a light-field camera.

kind of device is called a light field camera or a plenoptic camera. The sample of the plenoptic function it captures has 4 degrees of freedom: the angles (θ, ϕ) and also the position (x, y) where the light ray crossed the primary lens (assuming that the main lens is placed parallel to the $x \times y$ plane) [49–51].

In the case of point clouds, it is more common to capture this information using two or more cameras placed at different positions, capturing at the same time the light of a scene [52]. The image of the two cameras together offers samples of the light field, and this more abundant information enables us to estimate the missing parts of the light field and to gather information about the depth of the scene. We could improve the quality of the measure using even more cameras [53]. The main drawback of this framework is that the cameras need to be synchronized, precisely disposed to obtain an epipolar geometry, and need to have the same calibration.

Once the plenoptic function has been captured, it is straightforward to generate images by indexing the appropriate light rays. Also, the plenoptic function enables complex post-processing as refocusing [54–56] and depth estimation [51, 57, 58]. In our work, we are interested in the representation of the plenoptic function by means of point clouds, which are referred as plenoptic point clouds. From the captured scene, after post-processing, a sequence of points is generated representing the scene. In the case o plenoptic point clouds, each point is seen as a source of light, emitting the pencil of light in its position. Therefore, the pencil of light of each point is one of its attributes.

2.6 COMPARING THE PERFORMANCE OF COMPRESSION ALGORITHMS

When compressing an image, some artifacts are introduced on the reconstructed image, manly due to quantization. These artifacts are called noise. The lower the noise, the more accurate the reconstructe image is to the original one.

The performance of a compression algorithm is usually measured in terms of rate-distortion, where the rate is how many bits where used to encode such image while the distortion measures how noise have interfered in the reconstructed image quality.

The quality of a reconstructed image can be measured in terms of the peak signal-to-noise ratio (PSNR) that is the ratio between the maximum power of a signal (squared maximum amplitude) and the power of corrupting noise. PSNR is usually expressed in decibels (dB), a logarithmic scale, due to its wide dynamic range. Therefore, the PSNR is computed as

$$PSNR = 10\, log_{10}\left(\frac{A_{max}^2}{MSE}\right), \tag{2.22}$$

where A_{max} is the maximum signal amplitude and MSE is the mean squared error between the original and reconstructed signal. Higher values of PSNR correspond to higher quality of the reconstructed image, *i.e.*, a higher fidelity with the original image.

For point cloud compression, the PSNR can be evaluated using the signals from the Y, U or V channels, denoted by $PSNR_Y$, $PSNR_U$ and $PSNR_V$, and also the PSNR measured in all R, G, and B channels ($PSNR_{RGB}$) or even from a mixture of the signals such as $(6 * Y + U + V)/8$. However, all these PSNRs are highly correlated. The Pearson product-moment correlation coefficient [59] among these measures is shown in Table 2.1. It is apparent that the results are very highly correlated, and there is not much information to gain by repeating the presentation of results for all channels and color spaces.

Therefore, rate-distortion curves are usually only shown for Y channel for simplicity. Nevertheless, it reflects the performance with the other color channels.

Table 2.1: Correlation coefficient between $PSNR_Y$ for all point clouds in the 8i Labs dataset

	$PSNR_{RGB}$	$PSNR_U$	$PSNR_V$
maximum	0.9999	0.9990	0.9990
average	0.9996	0.9911	0.9922
minimum	0.9993	0.9807	0.9791

For comparing two rate-distortion curves as those shown in Figures 2.20 and 2.21, Gisle Bjøntegaard proposed a method and two metrics that were submitted to the Video Coding Experts Group in 2001 and latter adopted as a standard [60]. The two metrics are the:

- BD-PSNR: Represents the average PSNR difference between two curves for the same rate. In Figure 2.20, it is let the rate vary in a given range and is computed the integral of the

PSNR difference between the curves. From the area computed by the integral we compute the average difference by dividing it by the range of rate.

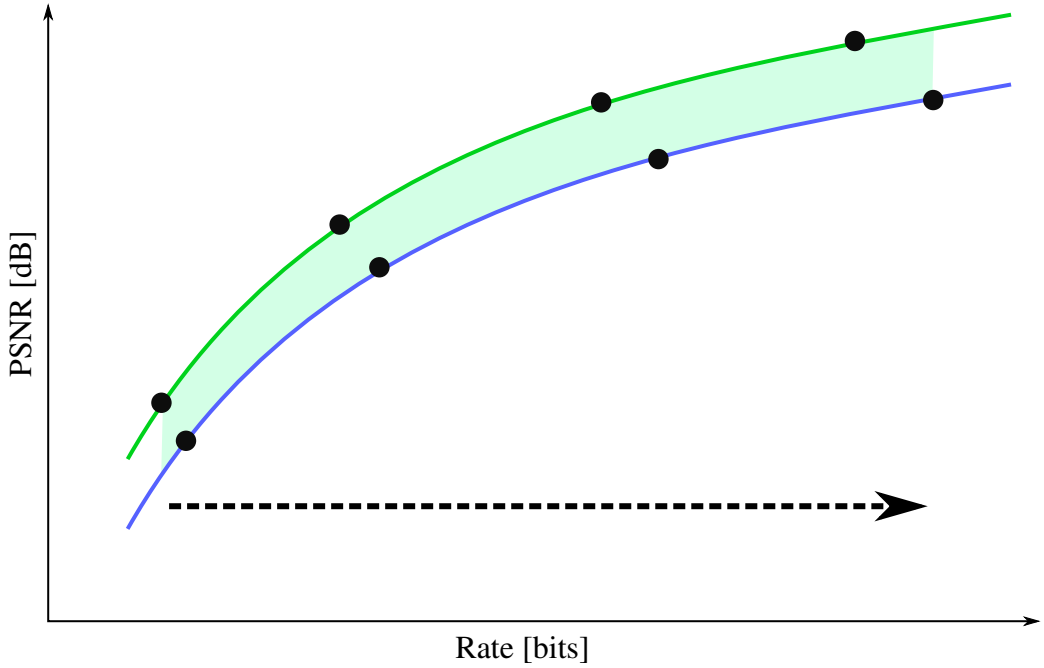


Figure 2.20: Computation of the BD-PSNR. 4 data poits for each curve are used to fit a curve representing the relationship between rate and PSNR. From the fitted curves we compute the average PSNR difference between them.

- BD-Rate: Represents the average rate difference between two curves for the same PSNR. In Figure 2.21, it is let the PSNR vary in a given range and is computed the integral of the rate difference between the curves. From the area computed by the integral we compute the average difference by dividing it by the range of PSNR.

Bjøntegaard proposed to used 4 data points for each curve, as shown in Figures 2.20 and 2.21, and fit a curve passing though these points. Bjøntegaard observed that, if the rate is given in logarithmic scale and the PSNR in dB, their relationship can be expressed by a third degree polynomial, *i.e.*:

$$PSNR = A_0 + A_1\ log(rate) + A_2\ log(rate)^2 + A_3\ log(rate)^3 \text{ or} \tag{2.23}$$

$$log(rate) = B_0 + B_1\ PSNR + B_2\ PSNR^2 + B_3\ PSNR^3, \tag{2.24}$$

where A_n and B_n are the parameters of the fitted curves.

Based on the fitted curves is computed the integrals of the difference PSNR or rate. The range of the rate or PSNR vary from the minimum to the maximum values found in these data points.

Both metrics can be used to compare two rate-distortion curves. For the BD-PSNR, one curve is considered better than the other if the BD-PSNR is positive and the higher the metric the better because this means that, on average, the PSNR is higher for a given rate. For the BD-Rate, on the

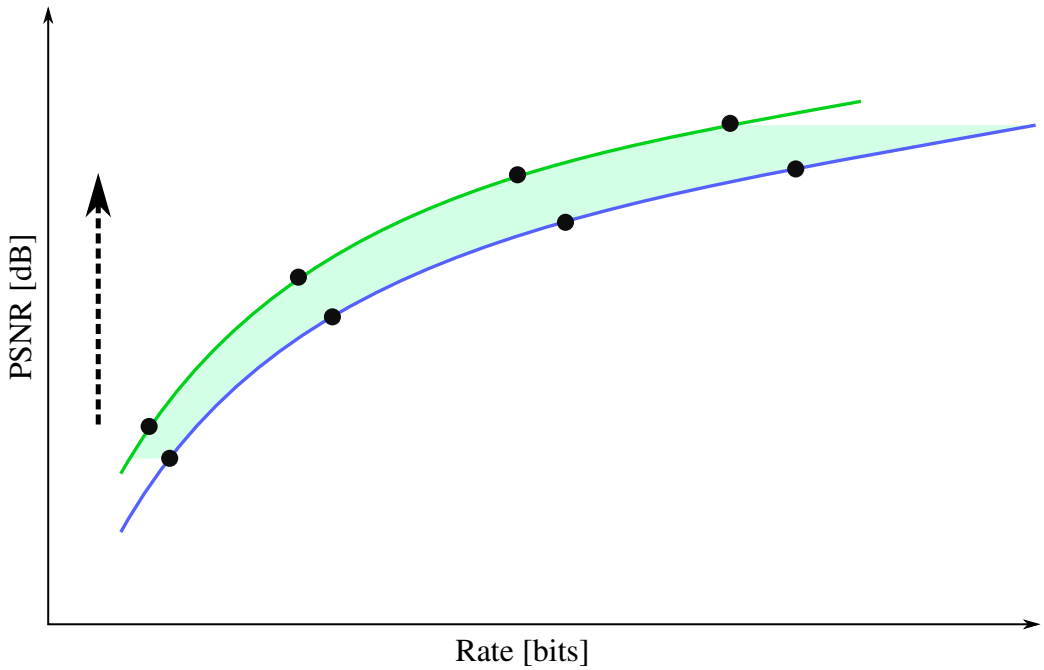


Figure 2.21: Computation of the BD-Rate. 4 data poits for each curve are used to fit a curve representing the relationship between rate and PSNR. From the fitted curves we compute the average Rate difference between them.

other hand, negative values correspond to a better performance and the lower the metric the better because this means that, on average, the algorithm requires fewer bits to encode the image with the same PSNR.

3 ENTROPY CODER

The RAHT is based in the wavelet transform with a Haar Kernel. Coefficients are quantized and entropy coded. In this chapter we focus on the entropy coder, more especifically in how to order the coefficients to improve the performance when using an adaptive coder. This work resulted in a paper published on arXiv on May 2018 [36].

The way coefficients are computed can be studied using a binary tree, as shown in Figure 3.1, where we have the voxel's attributes of the point cloud at the leaves of the tree (at level 4, in our example). Attributes are combined in pairs, producing inner nodes to the level below. We iterate over each level, caring coefficients to the level below until reaching the tree root.

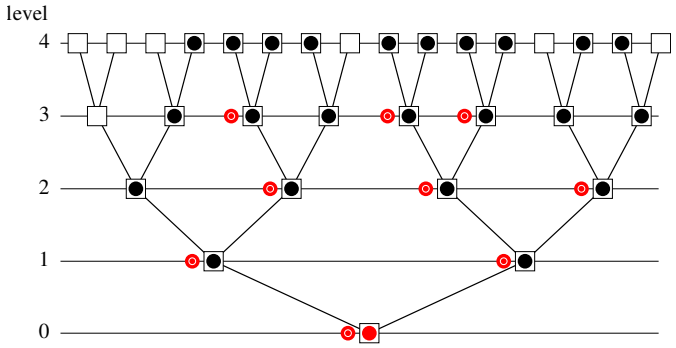
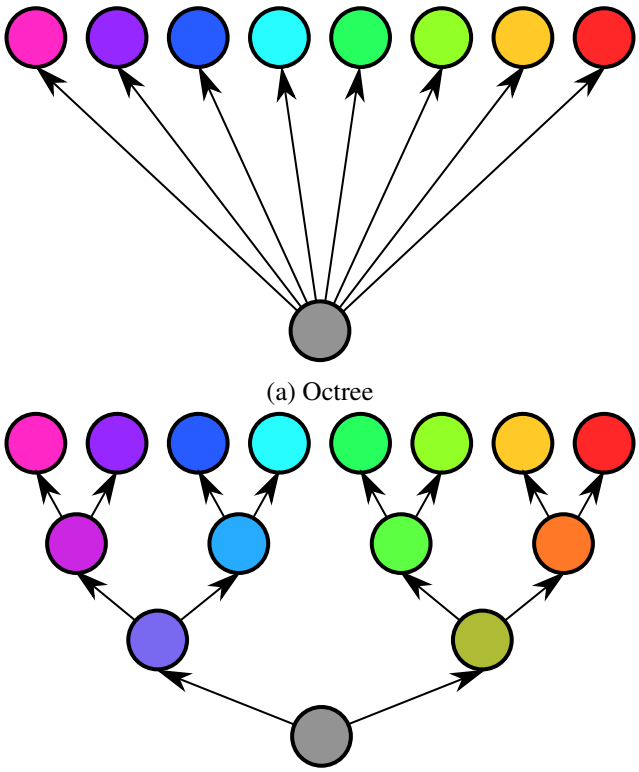


Figure 3.1: RAHT tree structure. This is a binary tree based in the octree. Every three levels in this tree correspond to one level of the octree.

How nodes and leaves are connected is dictated by the point cloud geometry, i.e., the octree. Even though the tree in Figure 3.1 is binary, it can be shown that an octree can be easily converted to a binary tree by introducing some intermediate steps as demonstrated in Figure 3.2. In this fashion, one level of the octree is translated to 3 levels in the binary tree.

One or both nodes in the pair being combined may be void of attributes. In this case, no high-pass coefficient is computed, and the attribute of the occupied node, if any, is promoted to the level below. On the other hand, when both nodes are occupied, theirs attributes are combined through a linear transformation given by Equation (2.17), producing a low-pass coefficient that will be the attribute of the node in the level below, and a high-pass coefficient. In Figure 3.1, low-pass coefficients are indicated by the symbol ●, while high-pass coefficients are indicated by the symbol ⊙. Every node in the tree has a weight. Occupied leaves have a weight equal to one, and void leaves have a weight equal to zero. Nodes resulting in a combination of two nodes have a weight equal to the sum of the combined nodes.

The decoder can compute the attributes of the point cloud by going trough this tree from root to leaves. As we assume that the decoder already has access to the point cloud geometry, sent using a geometry coder, it can reconstruct the tree structure. For the computation of the point cloud attributes is necessary to send the low-pass coefficient at the tree root and all high-pass co-



(a) Octree

(b) Binary tree

Figure 3.2: An octree can be converted to a binary tree by introducing some intermediate steps.

efficients produced during the tree descend in the encoder. These coefficients are encoded via an entropy coder based on arithmetic coding (AC). The results reported by Queiroz and Chou [29] demonstrate a performance which is comparable to that of the then state-of-the-art graph transform (GT) based coder [35], but at a much lower complexity. They also tested Adaptive run-length Golomb-Rice encoding (RLGR) [61], which is a much less complex entropy coder. Results were shown [29] that RLGR-based RAHT (or RLGR-RAHT) was noticeably inferior to AC-based RAHT (AC-RAHT), yet at a lower complexity.

RLGR is a general-purpose encoder that combines run-length and a Golomb-Rice encoder. It assumes that a Laplacian distribution with zero average can approximate the distribution of its input. This encoder has two parameters: the number of bits spend to encode the remainder for the Golomb-Rice and a threshold that establish a criterion to switch between run-length and Golomb-Rice modes. After each codeword is encoded, the encoder adapts its parameters with a backward rule. The decoder follows the same adaptation rules. Therefore there is no need to signal changes in parameters to the decoder.

In RLGR-RAHT, coefficients are vectorized in a one-dimensional sequence prior to entropy coding by searching the tree. Since RLGR is an adaptive encoder, how we search the coefficients may have an impact on performance. We want to show that by choosing the searching algorithm, one may improve the RLGR-RAHT performance to the point of outperforming not only RLGR-RAHT in [29] but also the AC-RAHT. With the RLGR-RAHT, we will show that its performance

improves over the state-of-the-art in point cloud compression at a reduced cost compared to AC-RAHT or GT-based coders [35, 41].

For AC-RAHT, coefficients are divided into buckets according to the weight of the node where they were generated. It is assumed a Laplacian distribution, whose variance is calculated for each bucket. The variances are then quantized and transmitted to the receiver for proper decoding. The encoder employs the quantized variances as the parameter of the Laplacian distribution, and coefficients are encoded using an arithmetic coder. As an arithmetic coder is applied, the order of the coefficients does not matter. In RLGR-RAHT, however, it is clear that changes in the order of coefficients may lead to different compression results as RLGR is an adaptive algorithm.

There exist numerous ways to search the coefficients in the tree. Of course, if we knew the color data, one would be able to sort the RAHT coefficients and achieve the best results. Still, the overhead for transmitting this information to the decoder might offset the benefits of increasing the zero-runs. We need only to use geometry information (which is available to both encoder and decoder) to estimate the color-derived coefficient amplitudes and to decide on the searching algorithm.

RAHT original implementation sends the DC component first. High-pass coefficients are scanned following the tree in a depth-first traversal order (Traversal RLGR-RAHT). Effectively it scans the coefficients from right to left of the tree as depicted in Figure 3.3. We can also scan the coefficients according to their depth, in a breadth-first scanning (BFS RLGR-RAHT), shown in Figure 3.4 or sorted by their weights (Weight RLGR-RAHT), shown in Figure 3.5. The idea of reordering the RAHT coefficients prior to the entropy coder is not new and was also proposed in [62]. In [62], RAHT coefficients were sorted by their weight in descending order and encoded using RLGR.

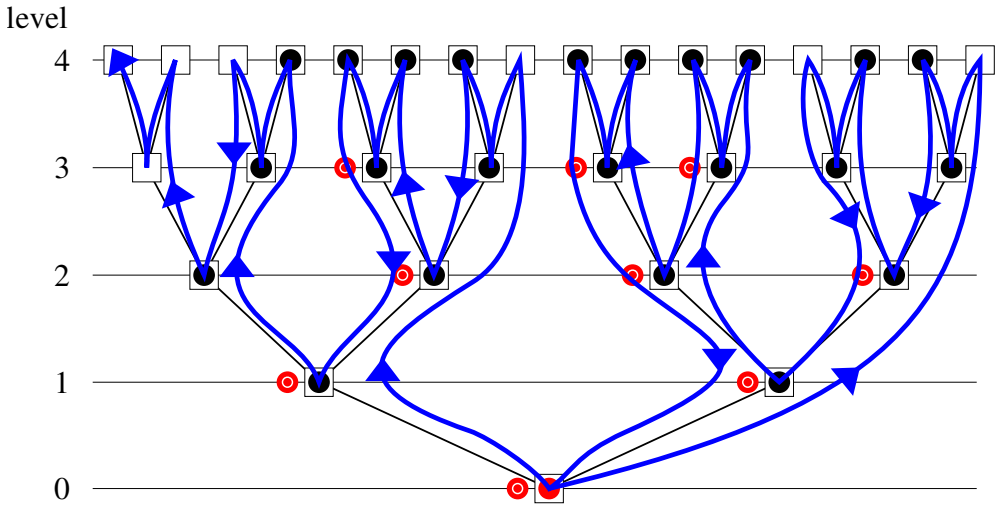


Figure 3.3: RAHT decomposition and ordering illustration for the depth-first traversal search.

The linear transformation of the nodes pair is similar to the Haar transform. Hence, RAHT coefficients generated further away from the root represent higher frequencies. For natural images, however, we expect the high-frequency components to have a lower amplitude than low-frequency

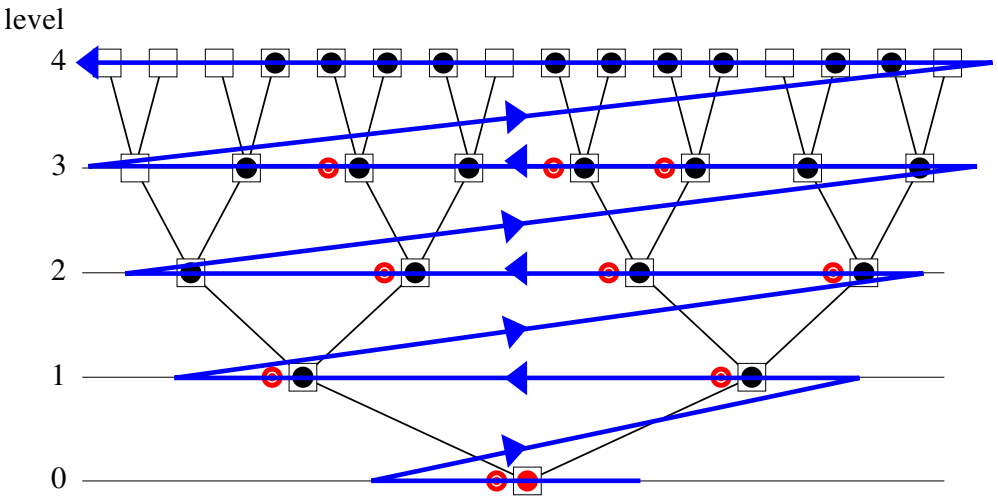


Figure 3.4: RAHT decomposition and ordering illustration for the breadth-first search.

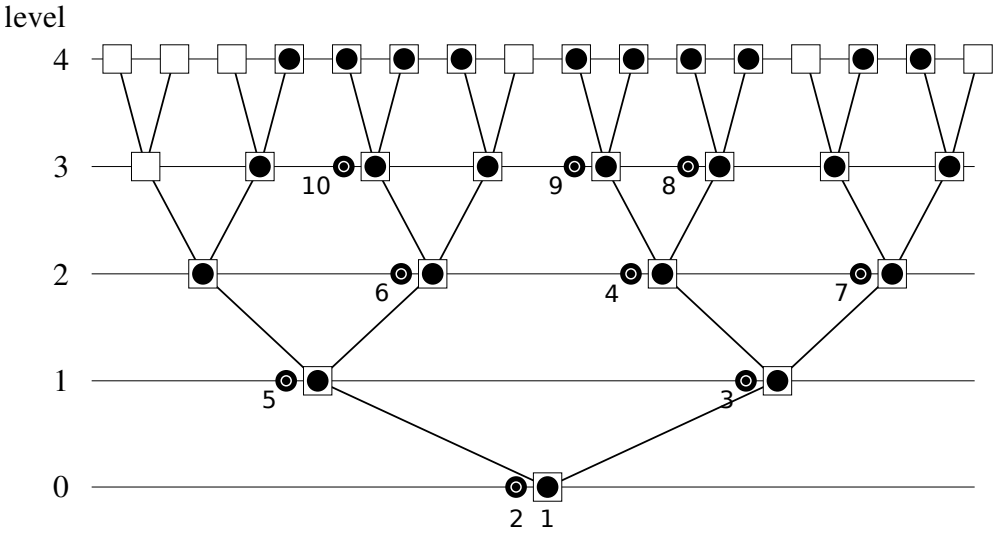


Figure 3.5: RAHT decomposition and ordering illustration for weight-sorted coefficients.

ones. Thus, quantizing high-frequency coefficients may lead to a more significant number of zero-valued quantized coefficients compared to low-frequency ones.

Equation (2.17) governs the linear transform in RAHT. If we assume that $F^{\ell+1}_{2i,j,k}$ and $F^{\ell+1}_{2i+1,j,k}$ have variances σ_1^2 and σ_2^2 and a mutual covariance γ, and $F^{\ell}_{i,j,k}$ and $G^{\ell}_{i,j,k}$ have variance σ_F^2 and σ_G^2, then

$$\sigma_F^2 = a^2\sigma_1^2 + b^2\sigma_2^2 + 2ab\gamma, \text{ and} \tag{3.1}$$

$$\sigma_G^2 = b^2\sigma_1^2 + a^2\sigma_2^2 - 2ab\gamma \tag{3.2}$$

The estimated variance can then be propagated down the tree, and for each level, we can compute the true covariance γ. If we remove the mean of YUV components beforehand, the estimated standard deviation of a coefficient can be used as an indication of its amplitude.

We then tested four sorting criteria to vectorize the coefficients:

1. **Traversal:** Depth-first traversal scanning of the binary tree;
2. **Breadth-first:** Breadth-first scanning of the binary tree;
3. **Weight:** Coefficients sorted by weight, from higher to lower weight;
4. **Variance:** Coefficients sorted by their estimated variance, from higher to lower.

The first one is the ordering in the original work. The latter 3 are our proposals. However, ordering by weight was also proposed by Chou *et. al.* [62]. We will then ascertain which sorting criteria would be more advantageous in Section 3.1.

3.1 EXPERIMENTAL RESULTS

To test the performance of the proposed sorting criteria we used the same seven test point clouds used in the work of Queiroz and Chou [29], named "Andrew", "Phil", "Ricardo", "Sarah", "Man", "Skier" and "Objects". The first 5 sequences are from the Microsoft voxelized upper bodies [63] (MVUB) dataset. Point clouds "Skier" is from the ITI database and "Objects" is from the 3DCOMET dataset [64].

Figure 3.6 compares the average zero-run-length when subject to RAHT and the given quantizer step for the four sorting criteria: weight, variance, breadth-first, and transversal. One can easily see that the breadth-first search yields longer zero-run-lengths sequences followed by sorting by weight as proposed by Chou *et. al.* [62]. Longer zero-run-length sequences reduce the number of bits necessary for encoding since the encoder only needs to convey the start of the run-length and how many coefficients are in the run. The variance sorting criteria yielded average zero-run-length sequences very similar to the weight sorting, while the traversal scanning, used in the original work, had the shorter sequences among the four tested criteria.

Furthermore, RLGR is an adaptive algorithm that continuously updates its parameters. Thus, it is expected that a more behaved signal that smoothly transitions between coefficients with high amplitude to ones with lower amplitude will need fewer bits to be encoded. Figures 3.7 and 3.8 compares the transversal and breadth-first search for frame "Phil", where we can expect that the better-behaved decaying pattern for the breadth-first case may lead to faster adaptation and better compression with the RLGR algorithm.

Tables 3.1 and 3.2 compare the four sorting criteria in RLGR-RAHT. In them and the rest of this work, the rate is given in bits per occupied voxel (bpov) and distortion in peak signal-to-noise ratio (PSNR in dB) comparing the luminance component of the original and reconstructed frames unless stated otherwise.

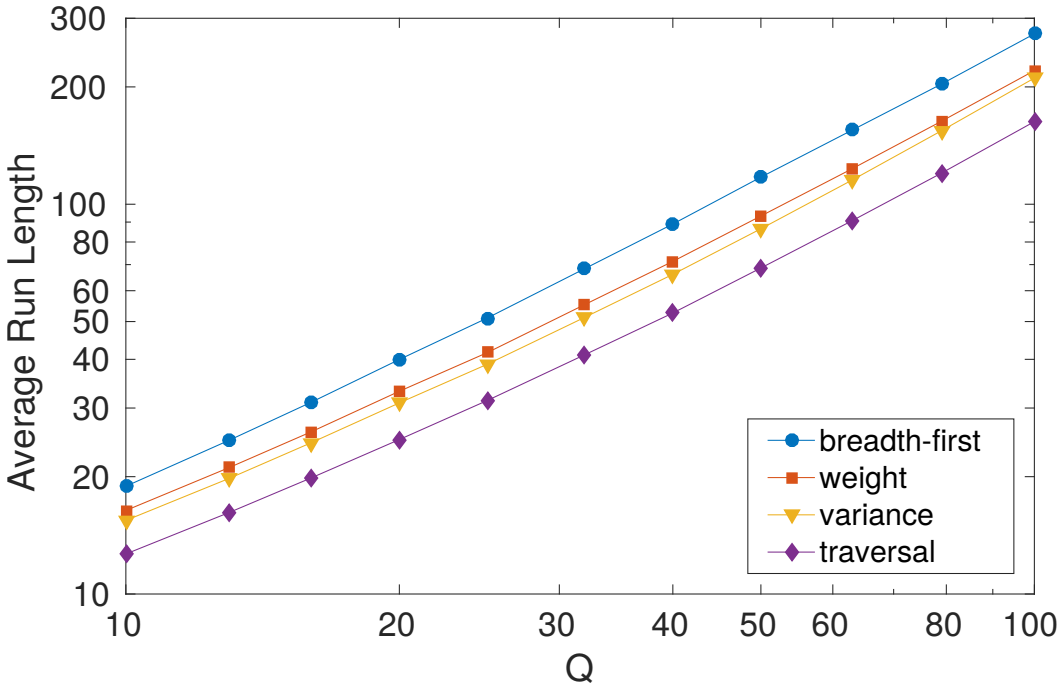


Figure 3.6: Average zero-run-length among the 7 test point clouds, using different sorting criteria. The higher the run-length the better as it requires fewer bits to be encoded.

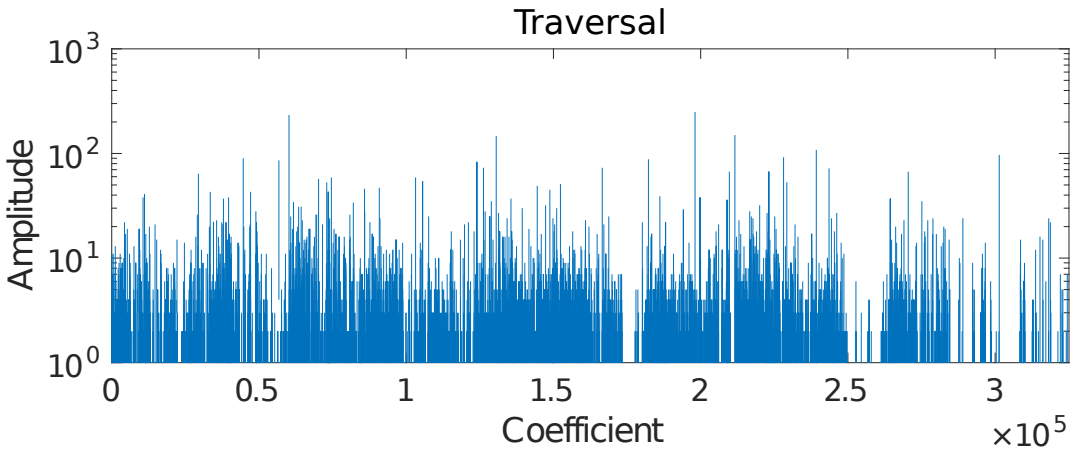


Figure 3.7: Quantized coefficient vectorialized with the traversal scanning for point cloud "Phil" ($Q_{step} = 20$). Coefficients present a random-like pattern.

We can see from Tables 3.1 and 3.2 that the proposed breadth-first sorting of the RAHT coefficients yields the best performance for all tested point clouds. We believe this result comes from the longer zero-run-length sequences that this sorting criteria produces for natural images, and also from the decaying pattern that induces a better adaption for the RLGR algorithm.

In light of this, we run the proposed breadth-first RLGR-RAHT for all images and quantizer steps in the previous work, and we believe that Figures 6, 7 and 8(a) in [29] should be replaced with the improved results in Figures 3.9 to 3.15 in this work.

From the many rate-distortion curves, it is clear that breadth-first RLGR-RAHT yields the

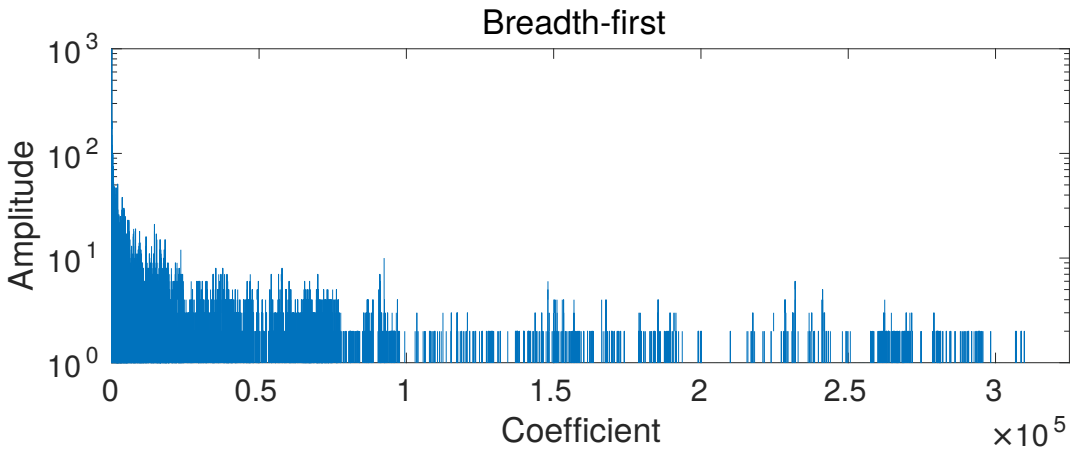


Figure 3.8: Quantized coefficient vectorialized with the breadth-first scanning for point cloud "Phil" ($Q_{step} = 20$). Coefficients present a somewhat decaying amplitude pattern.

Table 3.1: Rate comparison of the RLGR performance when using different sorting criteria ($Q_{step} = 10$).

	Bits per voxel				
Point cloud	**transversal**	**breadth-first**	**weight**	**variance**	**PSNR**
Man	3.1046	**2.0581**	2.1691	2.2145	40.3260
Andrew	4.4661	**3.3881**	3.4893	3.5149	39.8398
Phil	5.0985	**3.7117**	3.8018	3.8143	39.7470
Ricardo	1.5040	**0.8993**	1.0137	1.0023	43.6176
Sarah	1.9500	**1.1003**	1.2203	1.2422	43.4137
Skier	4.7303	**2.8717**	3.0303	3.0432	40.7751
Objects	4.2951	**3.3085**	3.4579	3.5029	39.9419

Table 3.2: Rate comparison of the RLGR performance when using different sorting criteria ($Q_{step} = 40$).

	Bits per voxel				
Point cloud	**transversal**	**breadth-first**	**weight**	**variance**	**PSNR**
Man	0.7638	**0.5345**	0.5570	0.5577	31.5615
Andrew	1.0225	**0.7437**	0.7715	0.7698	29.7742
Phil	1.0416	**0.6904**	0.7082	0.7090	30.6536
Ricardo	0.3395	**0.2074**	0.2254	0.2120	37.0116
Sarah	0.4474	**0.2455**	0.2772	0.2784	36.3494
Skier	1.3744	**0.9071**	0.9450	0.9366	31.6845
Objects	0.8551	**0.6692**	0.7043	0.6882	31.8801

best performance, even outperforming the GT-based coder.

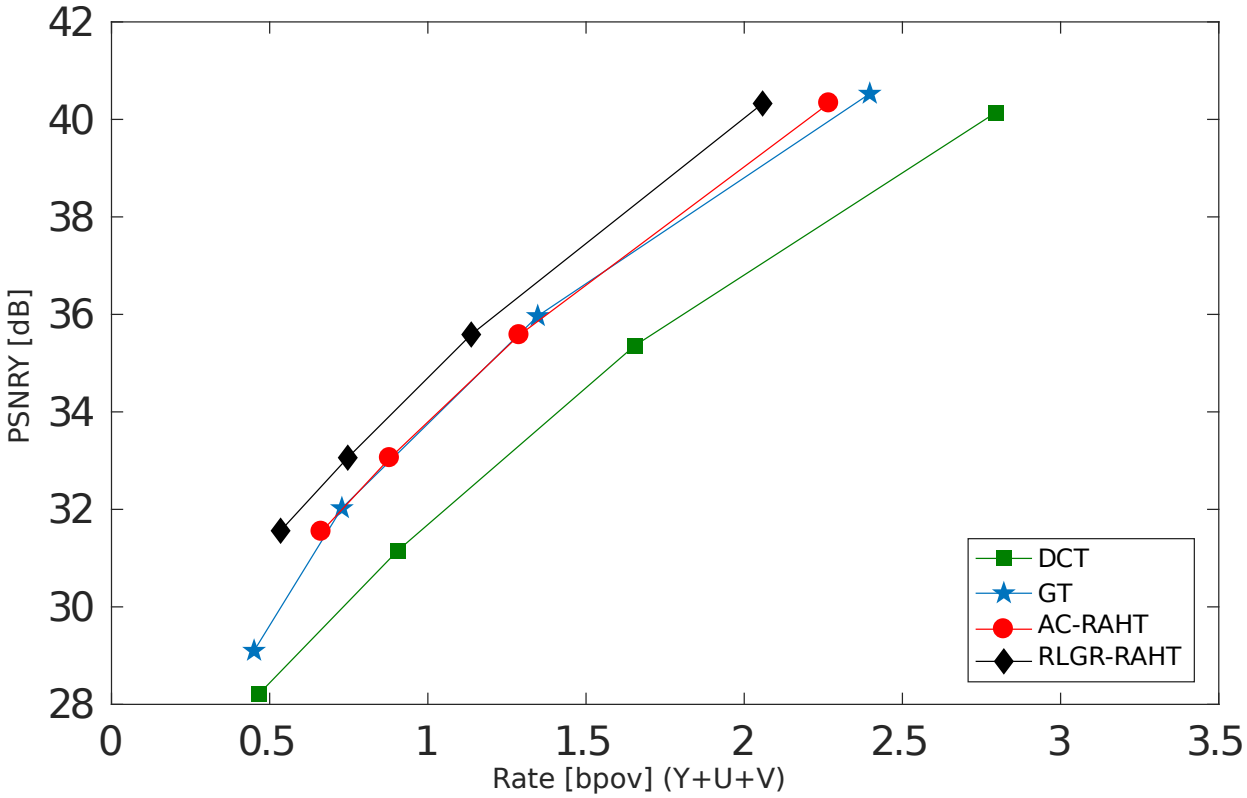


Figure 3.9: Rate-distortion curves for point cloud “Man” using RLGR-RAHT with the breadth-first sorting criteria.

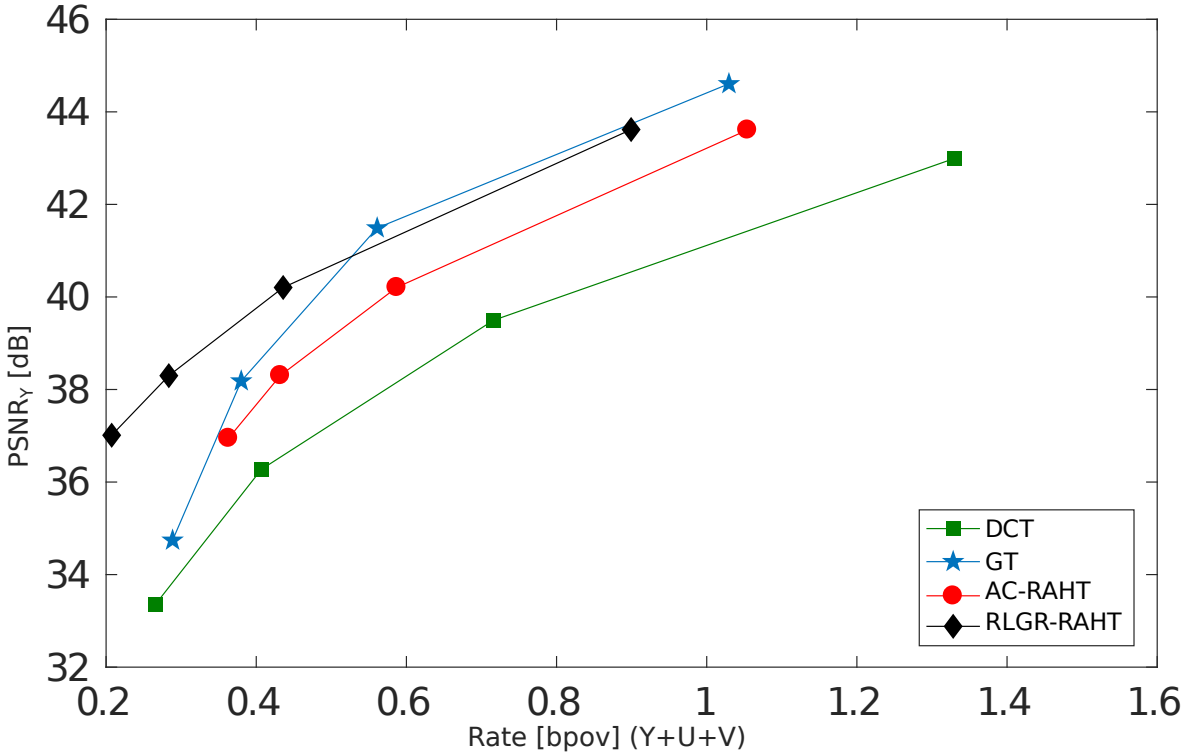


Figure 3.10: Rate-distortion curves for point cloud “Ricardo” using RLGR-RAHT with the breadth-first sorting criteria.

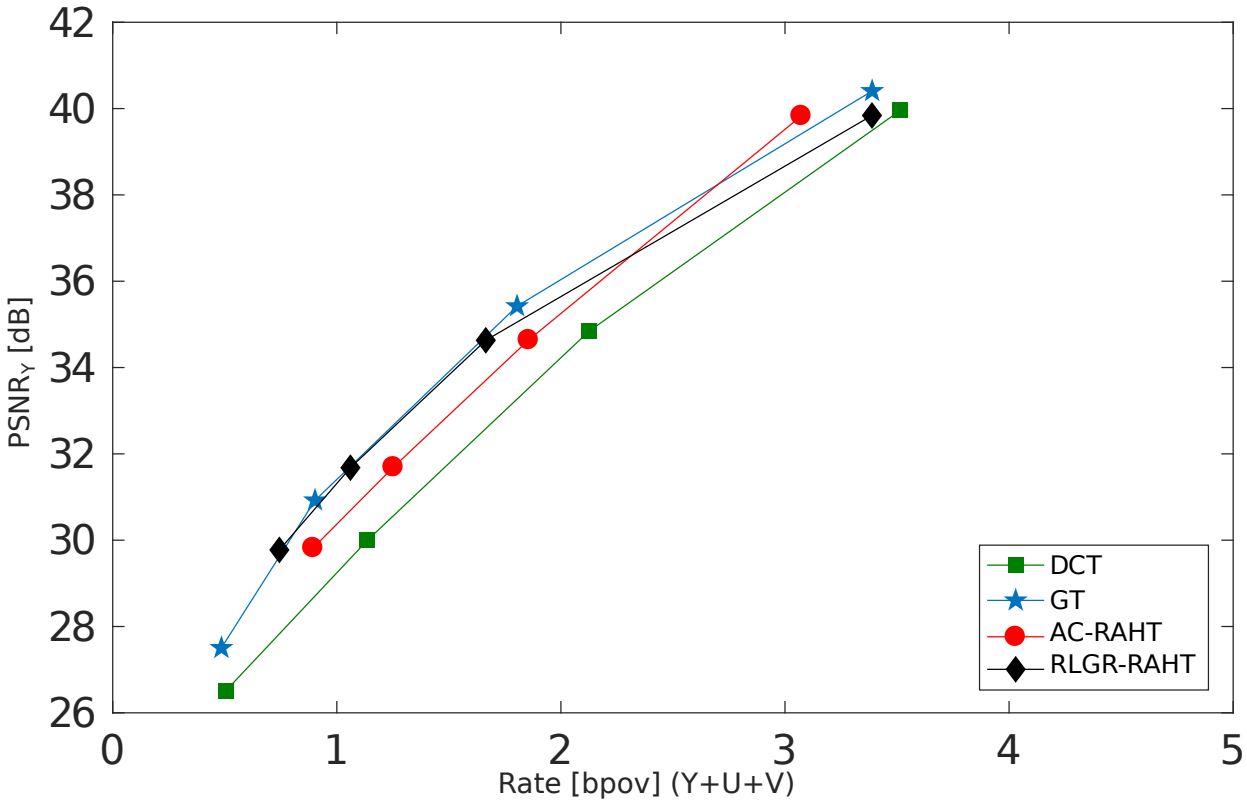


Figure 3.11: Rate-distortion curves for point cloud "Andrew" using RLGR-RAHT with the breadth-first sorting criteria.

3.2 CONCLUSION

We have presented and tested four sorting criteria to improve the performance of the RAHT entropy coder for 3D point cloud attributes. This transform has almost linear complexity and can be easily computed in real-time. The RLGR algorithm has linear complexity and is more straightforward than the arithmetic coder. The proposed method has a competitive performance, even outperforming the much more complex GT-based point cloud coder. We believe the proposed breadth-first RLGR-RAHT encoder presents the best performance so far in the compression of voxelized point clouds, with better rate-distortion trade-offs and lower complexity than rival approaches.

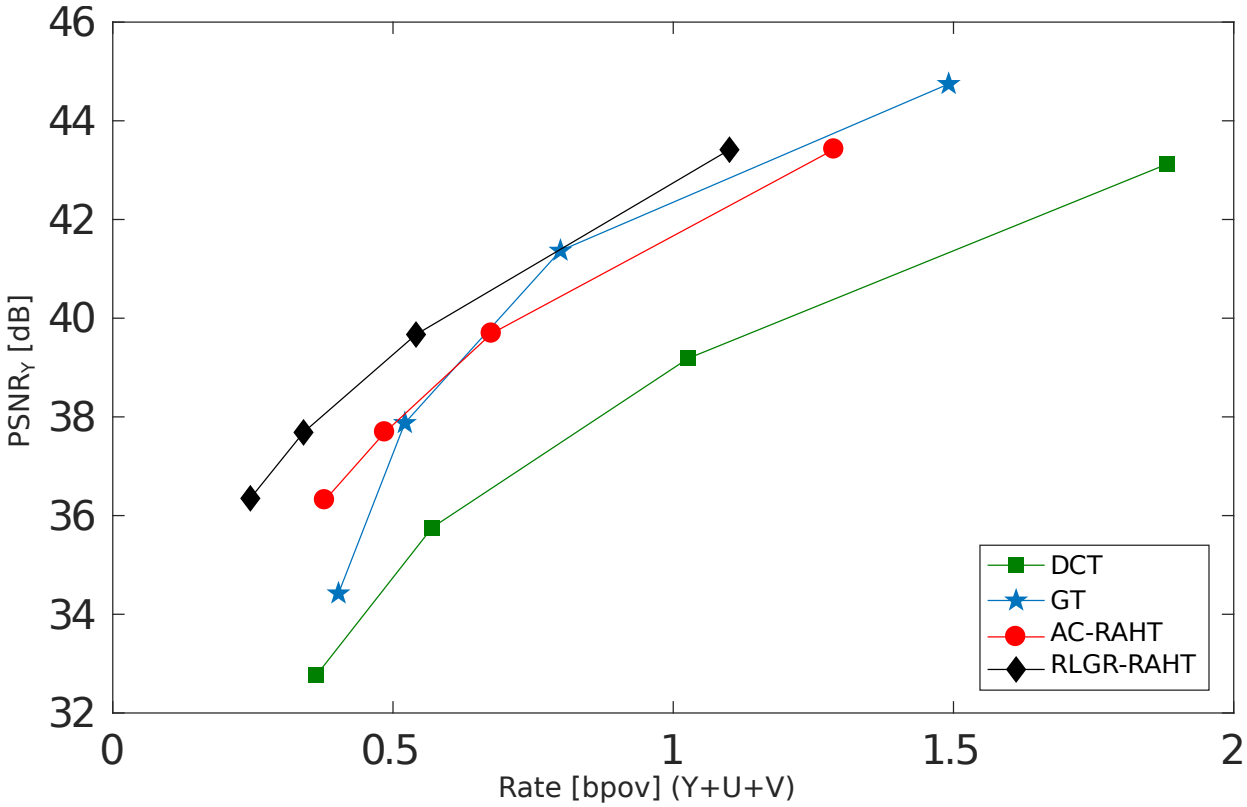


Figure 3.12: Rate-distortion curves for point cloud “Sarah” using RLGR-RAHT with the breadth-first sorting criteria.

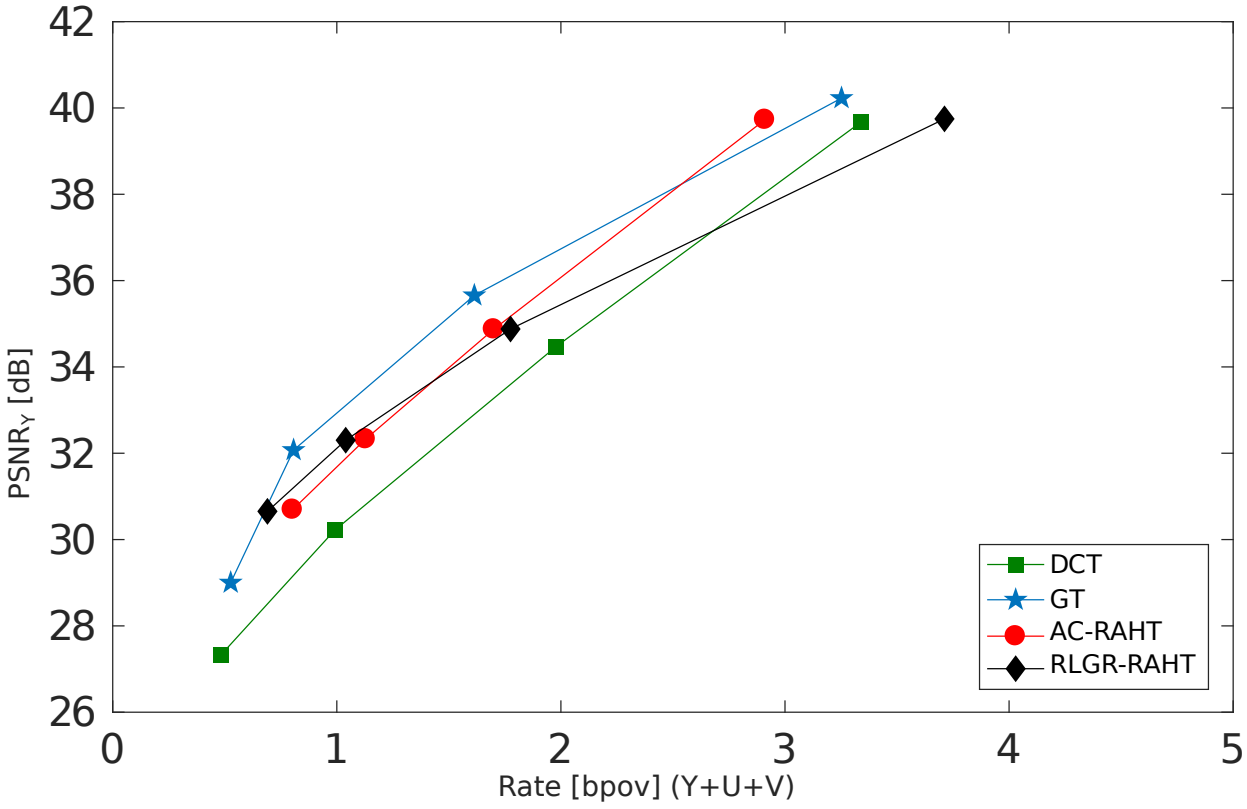


Figure 3.13: Rate-distortion curves for point cloud “Phil” using RLGR-RAHT with the breadth-first sorting criteria.

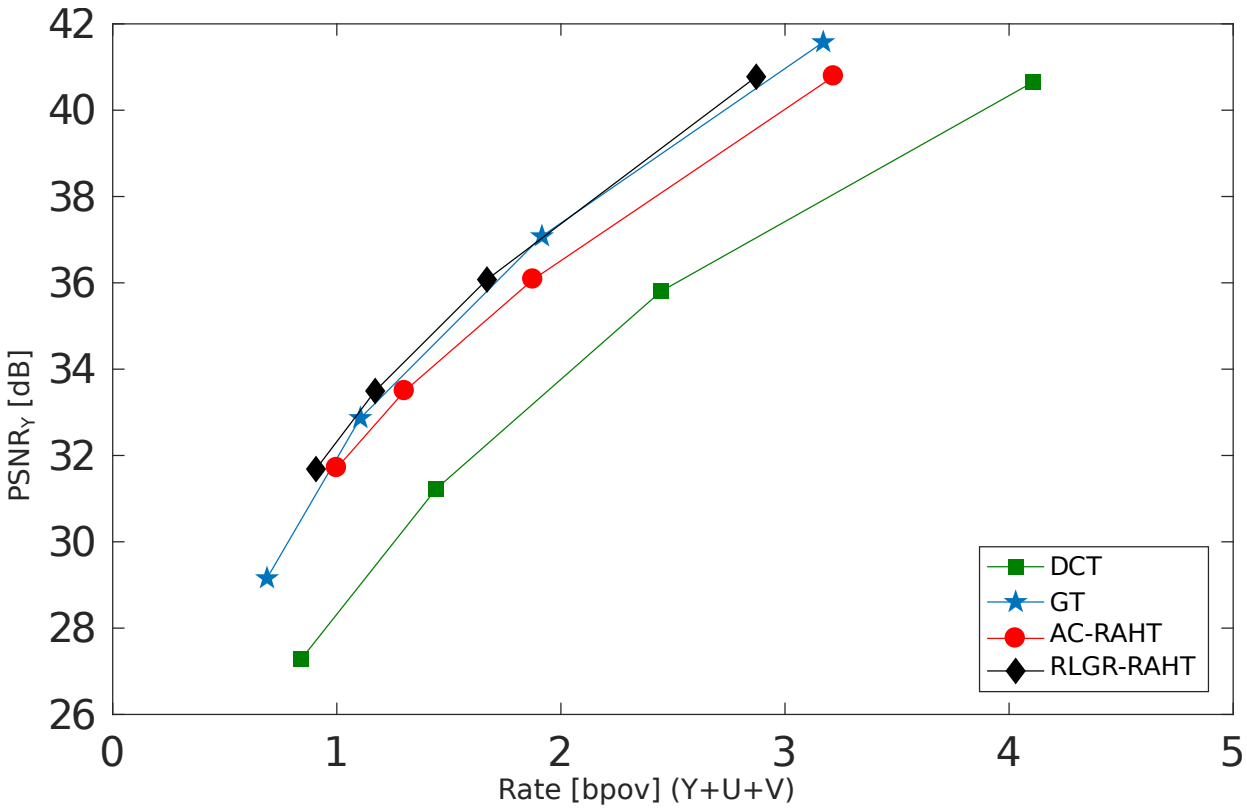


Figure 3.14: Rate-distortion curves for point cloud "Skier" using RLGR-RAHT with the breadth-first sorting criteria.

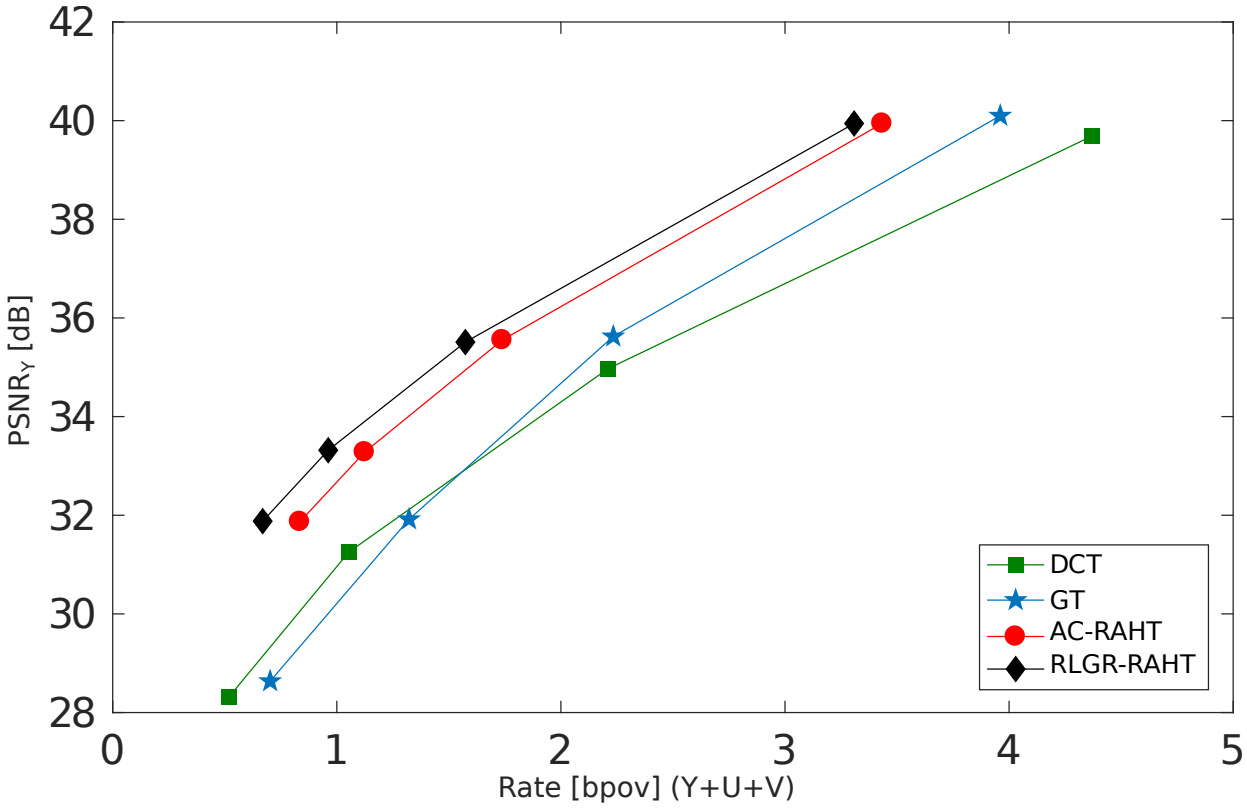


Figure 3.15: Rate-distortion curves for point cloud "Objects" using RLGR-RAHT with the breadth-first sorting criteria.

4 INTEGER TRANSFORM IMPLEMENTATION OF RAHT

RAHT is originally based upon a series of operations on real numbers that are performed using floating-point arithmetic. In standardization activities, recommendations should be as precise as possible to allow reproducibility. Even though floating-point arithmetic is well established nowadays through the IEEE 754 standard [65], we cannot guarantee that all machines implement this standard using the same requirements. To guarantee reproducibility, we present a novel RAHT description which is designed to accommodate fixed-point arithmetic. This chapter was published in a paper in the IEEE Signal Processing Letters [66]. It was also submitted to the International Organization for Standardization (ISO/ IEC JTC1/SC29/WG11) as an input document [67], that was latter adopted in the standard.

Fixed-point arithmetic can be performed using only integer values. Let us say, for example, that we want to represent the number 55.625. In binary, this number is written as 1100111.101_2. Multiplications of binary numbers by powers of 2 shifts the point. Therefore, this number could be also be expressed in binary as $1100111101_2 \times 2^{-3}$. Now, the value $1100111101_2 = 445_{10}$ is an integer value representing 55.625 with 3 bits of precision. More in general, we can represent a natural number a with fixed-point and β bits of precision as:

$$a = M_a \times 2^{-\beta}, \tag{4.1}$$

where M_a is an integer value. Let us assume two other numbers, $b = M_b \times 2^{-\beta}$ and $c = M_c \times 2^{-\beta}$, all with the same number of precision bits. The four basic mathematics operations can be performed as:

Operation		**Development**	**Integer equivalent**
Sum	$c = a + b$	$M_a \times 2^{-\beta} + M_b \times 2^{-\beta} = (M_a + M_b) \times 2^{-\beta}$	$M_c = M_a + M_b$
Subtraction	$c = a - b$	$M_a \times 2^{-\beta} - M_b \times 2^{-\beta} = (M_a - M_b) \times 2^{-\beta}$	$M_c = M_a - M_b$
Multiplication	$c = a.b$	$M_a \times 2^{-\beta}.M_b \times 2^{-\beta} = \left(\frac{M_a M_b}{2^\beta}\right) \times 2^{-\beta}$	$M_c = \frac{M_a M_b}{2^\beta}$
Division	$c = \frac{a}{b}$	$\frac{M_a \times 2^{-\beta}}{M_b \times 2^{-\beta}} = \left(\frac{M_a 2^\beta}{M_b}\right) \times 2^{-\beta}$	$M_c = \frac{M_a 2^\beta}{M_b}$

where a binary shift in integer arithmetic can perform the multiplication and division by powers of two.

When combining two voxels, the difference in weights is accounted for in RAHT as follows. Let two neighbor low-pass coefficients, $F^{\ell+1}_{2i,j,k}$ and $F^{\ell+1}_{2i+1,j,k}$ at level $\ell + 1$ be combined to form a low- and high-pass pair of coefficients, $F^{\ell}_{i,j,k}$ and $G^{\ell}_{i,j,k}$, at level ℓ. If $w^{\ell+1}_{2i,j,k}$ and $w^{\ell+1}_{2i+1,j,k}$ are the respective weights of the input coefficients, then

$$\begin{bmatrix} F^{\ell}_{i,j,k} \\ G^{\ell}_{i,j,k} \end{bmatrix} = \begin{bmatrix} a & b \\ -b & a \end{bmatrix} \begin{bmatrix} F^{\ell+1}_{2i,j,k} \\ F^{\ell+1}_{2i+1,j,k} \end{bmatrix} \tag{4.2}$$

where

$$a^2 = \frac{w_{\ell+1,2n}}{w_{\ell+1,2n} + w_{\ell+1,2n+1}},\ b^2 = \frac{w_{\ell+1,2n}}{w_{\ell+1,2n+1} + w_{\ell+1,2n+1}}. \tag{4.3}$$

Equation (4.2) is the same as presented in Equation (2.17). Note that $a^2 + b^2 = 1$ so that the transform is orthonormal, i.e. a Givens rotation [68, 69]. The typical *butterfly* used to describe its implementation is depicted in Figure 4.1.

Only the low-pass (F) coefficients are passed to further transforms. If we define, for simplicity of notation, $w_0 = w^{\ell+1}_{2i,j,k}$ and $w_1 = w^{\ell+1}_{2i+1,j,k}$, then

$$\begin{bmatrix} F^{\ell}_{i,j,k} \\ G^{\ell}_{i,j,k} \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{w_0}}{\sqrt{w_0+w_1}} & \frac{\sqrt{w_1}}{\sqrt{w_0+w_1}} \\ -\frac{\sqrt{w_1}}{\sqrt{w_0+w_1}} & \frac{\sqrt{w_0}}{\sqrt{w_0+w_1}} \end{bmatrix} \begin{bmatrix} F^{\ell+1}_{2i,j,k} \\ F^{\ell+1}_{2i+1,j,k} \end{bmatrix} \tag{4.4}$$

or

$$\begin{bmatrix} \frac{F^{\ell}_{i,j,k}}{\sqrt{w_0+w_1}} \\ G^{\ell}_{i,j,k} \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{w_0}}{w_0+w_1} & \frac{\sqrt{w_1}}{w_0+w_1} \\ -\frac{\sqrt{w_1}}{\sqrt{w_0+w_1}} & \frac{\sqrt{w_0}}{\sqrt{w_0+w_1}} \end{bmatrix} \begin{bmatrix} F^{\ell+1}_{2i,j,k} \\ F^{\ell+1}_{2i+1,j,k} \end{bmatrix} \tag{4.5}$$

or

$$\begin{bmatrix} \frac{F^{\ell}_{i,j,k}}{\sqrt{w_0+w_1}} \\ G^{\ell}_{i,j,k} \end{bmatrix} = \begin{bmatrix} \frac{w_0}{w_0+w_1} & \frac{w_1}{w_0+w_1} \\ -\frac{\sqrt{w_0 w_1}}{\sqrt{w_0+w_1}} & \frac{\sqrt{w_0 w_1}}{\sqrt{w_0+w_1}} \end{bmatrix} \begin{bmatrix} \frac{F^{\ell+1}_{2i,j,k}}{\sqrt{w_0}} \\ \frac{F^{\ell+1}_{2i+1,j,k}}{\sqrt{w_1}} \end{bmatrix} \tag{4.6}$$

or

$$\begin{bmatrix} \frac{F^{\ell}_{i,j,k}}{\sqrt{w_0+w_1}} \\ G^{\ell}_{i,j,k}\frac{\sqrt{w_0+w_1}}{\sqrt{w_0 w_1}} \end{bmatrix} = \begin{bmatrix} \frac{w_0}{w_0+w_1} & \frac{w_1}{w_0+w_1} \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \frac{F^{\ell+1}_{2i,j,k}}{\sqrt{w_0}} \\ \frac{F^{\ell+1}_{2i+1,j,k}}{\sqrt{w_1}} \end{bmatrix} \tag{4.7}$$

or

$$\begin{bmatrix} {F^{\ell}_{i,j,k}}' \\ {G^{\ell}_{i,j,k}}' \end{bmatrix} = \begin{bmatrix} a^2 & b^2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} {F^{\ell+1}_{2i,j,k}}' \\ {F^{\ell+1}_{2i+1,j,k}}' \end{bmatrix}. \tag{4.8}$$

Coefficients ${F^{\ell}_{i,j,k}}'$ and ${G^{\ell}_{i,j,k}}'$ have an embedded gain compared to the original $F^{\ell}_{i,j,k}$ and $G^{\ell}_{i,j,k}$:

$${F^{\ell}_{i,j,k}}' = \frac{F^{\ell}_{i,j,k}}{\sqrt{w_0 + w_1}},\quad {G^{\ell}_{i,j,k}}' = G^{\ell}_{i,j,k}\frac{\sqrt{w_0 + w_1}}{\sqrt{w_0 w_1}}. \tag{4.9}$$

Similarly, for the inverse,

$$\begin{bmatrix} {F^{\ell+1}_{2i,j,k}}' \\ {F^{\ell+1}_{2i+1,j,k}}' \end{bmatrix} = \begin{bmatrix} 1 & -b^2 \\ 1 & a^2 \end{bmatrix} \begin{bmatrix} {F^{\ell}_{i,j,k}}' \\ {G^{\ell}_{i,j,k}}' \end{bmatrix}. \tag{4.10}$$

If we consider that $a^2 = 1 - b^2$ and the following matrix decomposition [42]

$$\begin{bmatrix} a^2 & b^2 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & b^2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}, \tag{4.11}$$

then

$$\begin{bmatrix} {F^{\ell}_{i,j,k}}' \\ {G^{\ell}_{i,j,k}}' \end{bmatrix} = \begin{bmatrix} 1 & \Phi \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} {F^{\ell+1}_{2i,j,k}}' \\ {F^{\ell+1}_{2i+1,j,k}}' \end{bmatrix}, \tag{4.12}$$

which is implemented with two additions and one multiplication, and where Φ is an integer (fixed-point arithmetic) approximation of b^2, and $b^2 = w_1/(w_0 + w_1)$. The *butterfly* which implements the Givens rotation corresponding to each RAHT step is then simplified to the one depicted in Figure 4.1. The fixed-point approximation involves an integer division and a choice of the number of precision bits. The result is a fast transform without using floating-point arithmetic.

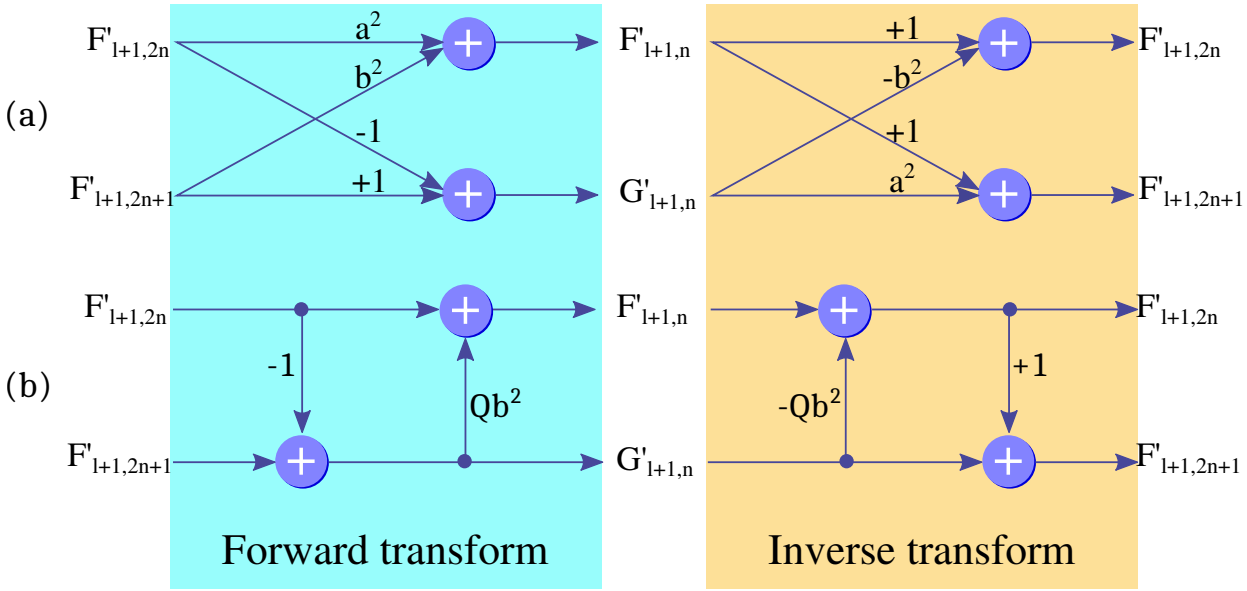


Figure 4.1: Lifting-style integer computation of the RAHT butterflies: (a) original butterfly for the forward and inverse transform; (b) the simplified fixed-point butterfly with its inverse counterpart.

There is a problem, though, with the proposed transform. Since there are scalings in both input and output samples, the transform itself is no longer orthonormal.

Fortunately, we can compensate them during quantization. If we assume, for simplicity of notation, a constant step size Δ for all levels and coefficients $F_{\ell,n}$ and $G_{\ell,n}$, the corrected stepsizes for the scaled coefficients should be

$${F^{\ell}_{i,j,k}}' \rightarrow \Delta \frac{1}{\sqrt{w_0 + w_1}}, \quad {G^{\ell}_{i,j,k}}' \rightarrow \Delta \frac{\sqrt{w_0 + w_1}}{\sqrt{w_0 w_1}}. \tag{4.13}$$

Recall that w_0 and w_1 are notational simplifications. If w_0 and w_1 are the weights of the input low-pass (F) coefficients, $w_0 + w_1$ is the weight of the output low-pass coefficient. Hence,

$$F_{i,j,k}^{\ell}{}' = \frac{F_{i,j,k}^{\ell}}{\sqrt{w_{2i,j,k}^{\ell+1} + w_{2i+1,j,k}^{\ell+1}}} = \frac{F_{i,j,k}^{\ell}}{\sqrt{w_{i,j,k}^{\ell}}}, \tag{4.14}$$

and the scaled output of one transform stage are at the proper scale to be input to the next stage. When we reach the root of the tree at level 0, the overall DC coefficient for the whole point cloud, F^0 is quantized using step size $\Delta/\sqrt{w^0}$. This requires us to compute one square root per point cloud. However, the high-pass coefficients, $G_{i,j,k}^{\ell}{}'$, are always quantized and encoded. For $G_{i,j,k}^{\ell}{}'$, the step size should be

$$\Delta \sqrt{\frac{w_{2i,j,k}^{\ell+1} + w_{2i+1,j,k}^{\ell+1}}{w_{2i,j,k}^{\ell+1} w_{2i+1,j,k}^{\ell+1}}}, \tag{4.15}$$

which demands the computation of a fixed-point square root per coefficient. Along with the square root, it also demands an addition, a division, and a multiplication, all using fixed-point arithmetic.

We need to compute as many fixed-point square roots as RAHT coefficients, which is the same number as occupied voxels in the point cloud. We proposed to employ the Newton–Raphson method to compute the fixed-point square root, which is a fast method to find the root of a function. The idea behind this method is to start with an initial guess, reasonably close to the true root, and then approximate the function by its tangent line. The intersection of the tangent line with the x-axis gives us a better approximation of the root based on the initial guess. The method then uses the new computed approximation as the initial guess for the next iteration. Iterations are repeated until reaching the actual root or until reaching the desired precision (see Figure 4.2).

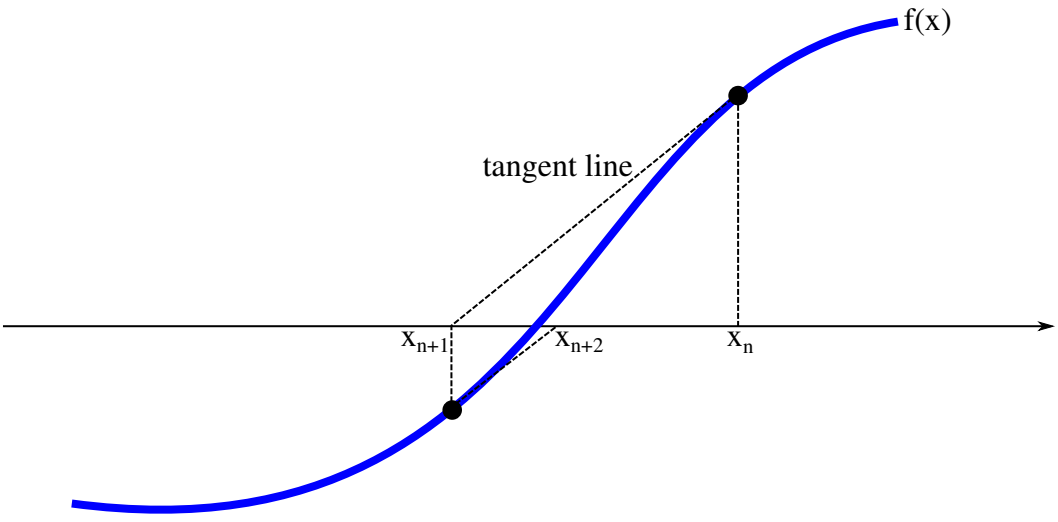


Figure 4.2: Newton-Raphson method to interactively find the root of a function. x_n is the initial guess at the n-th iteration and x_{n+1} is the new estimated root from the initial guess.

The update rule to compute the root can be derived from calculus as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \tag{4.16}$$

where x_n is the initial guess at the n-th iteration, x_{n+1} is the new computed root, $f(x)$ is the function we want to find its root and $f'(x)$ its derivative with respect to x.

Let us assume we want to compute the square root of the value v, given by $x = \sqrt{v}$. It is an equivalent problem as finding the root of the function $f(x) = x^2 - v$. Using Newton-Raphson method we get:

$$\begin{aligned} x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)} \\ &= x_n - \frac{x_n^2 - v}{2x_n} \\ &= x_n - \frac{x_n}{2} + \frac{v}{2x_n} \\ &= \frac{x_n}{2} + \frac{v}{2x_n} \\ &= \frac{x_n + v/x_n}{2} \end{aligned}$$

If x_n and v are represented in fixed-point by the integers X_n and V, as $x_n = X_n \times 2^{-\beta}$ and $v = V \times 2^{-2\beta}$, where $\sqrt{v} = \sqrt{V \times 2^{-2\beta}} = \sqrt{V} \times 2^{-\beta}$, then the update rule can be written using only integer arithmetic:

$$X_{n+1} = \left(X_n + \frac{V}{X_n} \right) \text{»} 1, \tag{4.17}$$

where $\bullet\text{»}b$ represent a binary shift equivalent to dividing by 2^b using integer arithmetic, while $\bullet\text{«}b$ is a binary shift equivalent to multiply by 2^b. Remark that the value v was written with double the precision of x_n when represented with fixed-point. This was done so Equation (4.17) would not require an additional binary shift.

The number of iterations the Newton-Raphson method needs to do to obtain the desired precision depends on the initial guess. It can be reduced using a look-up table with pre-computed squared roots. This look-up table cannot include all possible values because this would require an enormous amount of memory, but should be big enough to provide initial guesses close to the real root. We decided to pre-compute 256 squared root values for when $0 \leq V < 256$. This table can be used to produce initial guesses for any value V if we consider that

$$\sqrt{V} \approx \sqrt{\frac{V}{2^b} 2^b} = 2^{b/2} \sqrt{\frac{V}{2^b}} = \sqrt{V\text{»}b} \text{ « } (b/2) \tag{4.18}$$

and we chose b to be an even number respecting the condition $0 \leq V\text{»}b < 256$.

Equation (4.18) is not the exact value of $\sqrt{V}$ but an approximation. We are assuming that all operations are performed with integer arithmetic that introduces some inaccuracies in the division.

Therefore, our initial guess is

$$X_0 = \sqrt{V \gg b} \ll (b/2), \tag{4.19}$$

with $\sqrt{V \gg b}$ given by the look-up table.

The value of b was chosen to be:

$$b = \begin{cases} 56, & \text{if } 2^{56} \leq V < 2^{64} \\ 48, & \text{if } 2^{48} \leq V < 2^{56} \\ 40, & \text{if } 2^{40} \leq V < 2^{48} \\ 32, & \text{if } 2^{32} \leq V < 2^{40} \\ 24, & \text{if } 2^{24} \leq V < 2^{32} \\ 16, & \text{if } 2^{16} \leq V < 2^{24} \\ 8, & \text{if } 2^{8} \leq V < 2^{16} \\ 0, & \text{if } V < 2^{8} \end{cases} \tag{4.20}$$

We have observed that with this initial guess and only two iterations of the Newton-Raphson method, there is a negligible difference in performance between RAHT implemented with floating-point arithmetic and our fixed-point implementation. Experimental results are given in the next section.

4.1 EXPERIMENTAL RESULTS

We compared our fixed-point RAHT implementation to the floating-point one, within the context of a RAHT-based point cloud coder. The idea is to measure the effect of the change on the coder performance in terms of rate-distortion (RD) curves. Rate is computed as the number of bits to encode all YUV color components, while distortion is measured as peak signal-to-noise ratio (PSNR) comparing the original and reconstructed Y-channel attributes. In our tests, we used the same number of precision bits for both Φ and the square root computation.

In addition to the point clouds used for testing in chapter 3, we also used the voxelized full bodies point clouds dataset from 8i Labs [70] (8iVFB).

Rate-distortion curves are shown in Figure 4.3, comparing the floating-point implementation against integer implementations with varying precision. The two sets of curves are very similar and show the curve for an 8-bit implementation to superimpose the floating-point one. The curve sets for the other point clouds were omitted because, qualitatively, they are the same as those two. To better appreciate how close the curves are, the average PSNR difference between fixed-point curves and the floating-point one was computed for every point cloud, and the results are shown

in Table 4.1.

Table 4.1: Average PSNR difference between fixed- and floating-point RAHT coder implementations. The average was computed in the rate range from 0.3 to 3 bits per occupied voxels. Precision of the fixed-point implementation is indicated.

Precision	**andrew**	**david**	**phil**	**ricardo**	**sarah**	**longdress**	**loot**	**redandblack**	**soldier**
1 bit	8.200	8.571	8.251	8.602	9.252	7.832	8.559	7.771	8.189
2 bits	2.685	2.561	2.615	2.671	2.856	2.518	2.681	2.405	2.622
3 bits	0.998	1.001	0.952	1.009	1.106	0.937	0.977	0.855	0.947
4 bits	0.430	0.497	0.414	0.472	0.481	0.390	0.424	0.348	0.391
5 bits	0.193	0.170	0.174	0.210	0.234	0.179	0.186	0.158	0.173
6 bits	0.095	0.081	0.085	0.076	0.098	0.084	0.080	0.074	0.083
7 bits	0.044	0.043	0.040	0.043	0.058	0.042	0.039	0.036	0.040
8 bits	0.022	0.027	0.021	0.030	0.037	0.020	0.018	0.018	0.016
9 bits	0.014	0.016	0.013	0.019	0.030	0.010	0.012	0.009	0.010
10 bits	0.007	0.013	0.007	0.016	0.025	0.005	0.007	0.005	0.005
11 bits	0.004	0.018	0.005	0.016	0.039	0.002	0.005	0.003	0.003

4.2 CONCLUSION

We have introduced a new RAHT definition that allows for fixed-point (integer) implementation without impacting the compression performance for applications in real-time point cloud transmission (*e.g.*, augmented reality, autonomous navigation). This result can effectively change the RAHT coder description in the context of point cloud compression standardization, and it may enable more players in the area. Results show that as low as 8-bit fixed-point precision may be good enough for our compression applications, and any representation using 10 bits and above yields negligible compression performance impact.

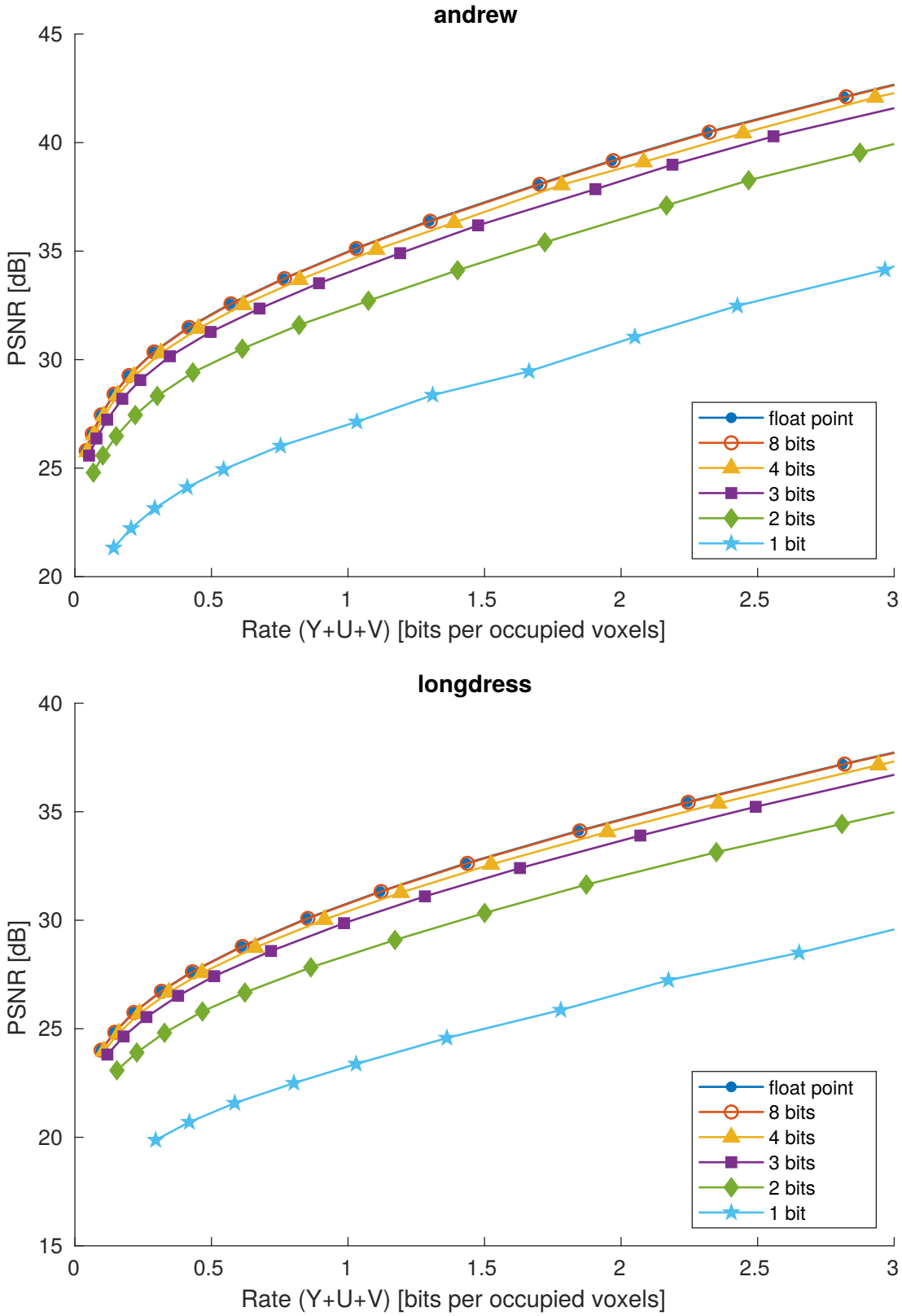


Figure 4.3: PSNR curves comparing the floating-point implementation with the fixed-point ones for two point clouds and different numbers of precision bits. Qualitatively, all sets of curves for all nine point clouds in our test set are very similar.

5 PLENOPTIC POINT CLOUDS

Point clouds are the result of processing all the information captured by an array of cameras combined with depth maps [71, 72] or even from light-field cameras [73]. Most works neglect the fact that a given point in space may change color according to the viewing angle and attribute just one RGB color triplet to a voxel [24–26, 29, 35, 74, 75]. This is a good approximation if the objects they are trying to represent act as a diffuse source of light, but tends to be less realistic when specular surfaces are at play where the color of a given point significantly varies according to the view direction (See Figure 5.1). The extreme case is a mirror, for which the emitted color is a reflection of its environment.

Figure 5.1: Three viewpoints of the same scene to highlight the color variation according to viewing direction. The colors at the same point on the wet floor or the car surface vary when the viewing point changes.

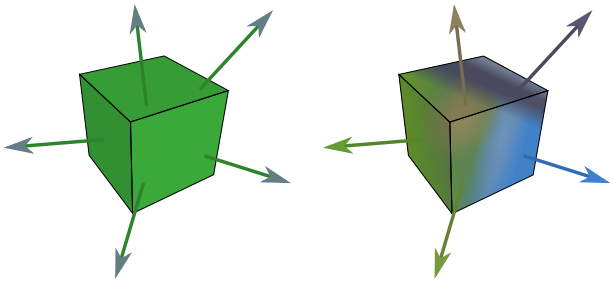
Figure 5.2: The mirror is an extreme case of a specular surface, whose emitted color changes completely according to the viewing angle. This image was created from a combination of images found on the internet.

In this chapter, we extend the representation of point clouds to allow for a voxel to change color according to the viewing angle and we propose how to compress such data. This chapter resulted in a paper published on the IEEE Transactions on Image Processing [76], a paper in the International Conference on Image Processing [77], a paper in the XXXVI Brazilian Symposium on Telecommunications and Signal Processing [78] and two input documents submitted to the

International Organization for Standardization (ISO/ IEC JTC1/SC29/WG11) [79, 80].

The plenoptic function is idealized in computer vision and describes how light is emitted in every point in space in any direction. We can consider that every camera in the array registering the point cloud is capturing a sample of the plenoptic function, and, in this sense, point clouds can be used to provide a better representation of the plenoptic function. In such, we can imagine that every point, or voxel, is a source of light, and the light it emits varies with direction.

Then, we define two types of voxels: non-plenoptic voxels are those that assume the objects are wholly diffuse and emit the same color in all directions; plenoptic voxels, on the other hand, preserve the information of how light changes with direction.



(a) Non-plenoptic voxel (b) Plenoptic voxel

Figure 5.3: A non-plenoptic voxel has no directional color information. This information is present in a plenoptic voxel and can be used to represent a scene in a more realistic way.

Most works in this area try to compress the light field of a scene using a multi-view or lenslet representation. The light field, whose definition is sometimes confused with the definition of the plenoptic function, is a vector function describing the amount of light flowing in every direction and every point. The plenoptic function is the space of all possible light rays passing through a point in space. A multi-view representation is essentially a collection of images captured from different viewpoints, while a lenslet representation is a collection of images representing how each point appears from different directions.

In multi-view methods, there is much redundancy among the images captured from different perspectives that can be exploited by inter-view prediction. Some methods that rely on techniques for inter-frame prediction have achieved remarkable improvements [81–85].

Methods based in the lenslet representation exploit intra-image similarities and directly compress the lenslet image [86–91] using techniques similar to intra-block prediction used in modern video compression.

However, these light field compression do not exploit geometric information directly. More significantly, extrapolation of views to those away the captured ones may be difficult due to occlusions, for example, and make these methods not very practical for virtual or augmented reality.

The use of point clouds eliminates the occlusion problem in view syntheses. Also, if we are capable of making a point, or voxel, in the point cloud change color according to the perspective, we have a rich representation of the plenoptic function. In this work, we associate with each voxel

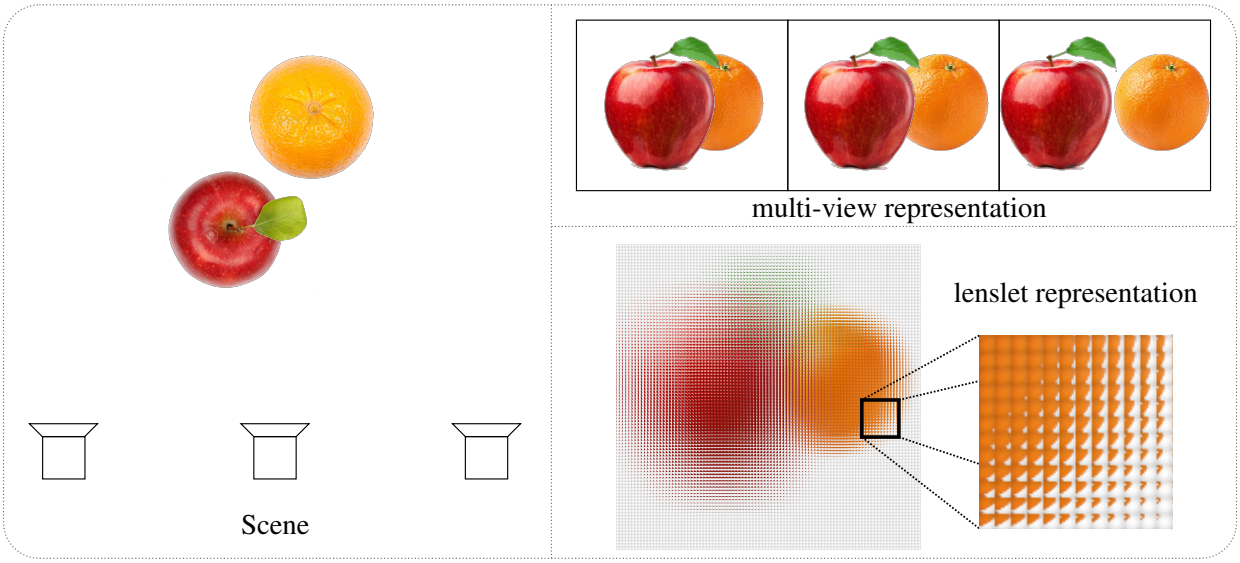


Figure 5.4: Multi-view versus lenslet representation. In the multi-view representation we have a collection of images captured from different perspectives, while in the lenslet representation we have a collection of how each point is seen from different perspectives.

the colors seen by all cameras. Since cameras are placed at different positions, if we know the camera and voxel positions, one can calculate samples of the plenoptic function for each voxel. We refer to this representation as the plenoptic point cloud since it is based on the plenoptic function. For other directions beyond those sampled (captured), we can synthesize the color by interpolation.

The main difference compared to a non-plenoptic point cloud is the number of attributes. Now we have one color triplet for each camera associated with each voxel. While we could encode this data assuming that it just has more attributes, this is inefficient since we are not taking into account the redundancy among cameras. In this chapter, we want to address the problem of encoding plenoptic point clouds. We have tested two approaches in order to extend RAHT to accommodate plenoptic (multiple colors) point clouds:

- The first one is described in Section 5.1 and consists of first applying an intermediate transform to reduce the redundancy among cameras, and then the attributes are encoded using RAHT. In this way, the coefficients of the transform are the attributes of the point cloud that are encoded by RAHT.

- The second approach is described in Section 5.2. We modify the geometry of the PC, increasing the number of voxels. Voxels are divided into sub-voxels. Each subvoxel has the information of just one camera, and its position indicates to which voxel and camera it belongs. Efficiently, this equivalent to enlarging the geometric resolution of the point cloud and the original geometry can be obtained through downsampling. This PC of sub-voxels is identical to a non-plenoptic point cloud, and we can employ RAHT without further modification.

The approaches that best resemble the ones we proposed here use a surface light field representation [92–94]. In those, they first determine the parameters of a function describing how colors change with direction and encode a point cloud with these parameters as attributes. The difference to our approach is conceptual. While in our approach, it is left to the decoder to interpolate the data to obtain the color in any desired perspective, with the surface light field, we can argue that the interpolation is performed at the encoder.

5.1 COLOR VECTOR TRANSFORMS

A plenoptic point cloud comprises a list of voxels $\{\nu_i\}$, each being described by its geometry (location in space) and color (in RGB or YUV color spaces) seen by the N_c cameras, *i.e*,

$$\nu_i = \left[x_i, y_i, z_i, R_i^1, G_i^1, B_i^1, \cdots, R_i^{N_c}, G_i^{N_c}, B_i^{N_c}\right]. \tag{5.1}$$

The color vector $\left[R^1, G^1, B^1, \cdots, R^{N_c}, G^{N_c}, B^{N_c}\right]$, is referred as the plenoptic appearance attribute of a voxel. See Figure 5.3. It usually contains a high level of redundancy that can be exploited for compression. It can be compared to 2D images where neighboring pixels are usually highly redundant, as well. Most compression algorithms for 2D images apply a transform to provide energy compaction to enable compression. Hence, we first apply a transform to the plenoptic appearance color vector (one transform for each color component, independently). For each component, it generates $N_c - 1$ high-pass coefficients and one DC coefficient that represents the average color of a voxel.

In this Section, we exploit 1D and 2D transforms to try to eliminate the redundancy of the color vector. The camera displacement can be used to estimate how correlated cameras are. We expect that cameras correlation decreases as the viewing angle difference between then increase. This is exploited in 2D in Section 5.1.1, where we project the viewing angle of each camera to a position in a 2D map. At this position we associate the color as seen by that camera, thus producing an image. This image is naturally sparse as not all viewing angles are captured. Then, we apply a 2D transform to this image. This transform produces coefficients that are set as the attributes of each voxel of the point cloud. Compression is acquired by the decorrelation of cameras.

In Section 5.1.2, we raster the cameras to a 1D vector and then apply a 1D transform. One natural choice is to estimate the covariance matrix between cameras to determine the optimum transform that decorrelates them. Another approach is to order the cameras in this vector, trying to preserve their neighborhood and use the well-known discrete cosine transform. This approach is sub-optimal but eliminates the need to send side information as both encoder and decoder are aware of the transform.

5.1.1 Cameras 2-dimensional projection to a θh plane

Let us assume a tiny sphere surrounding a voxel, viewed by several cameras, as illustrated in Figure 5.5. Each camera may be observing different colors as the light that the sphere emits may vary with each direction. These cameras are capturing a sample of the plenoptic function at this particular point in space.

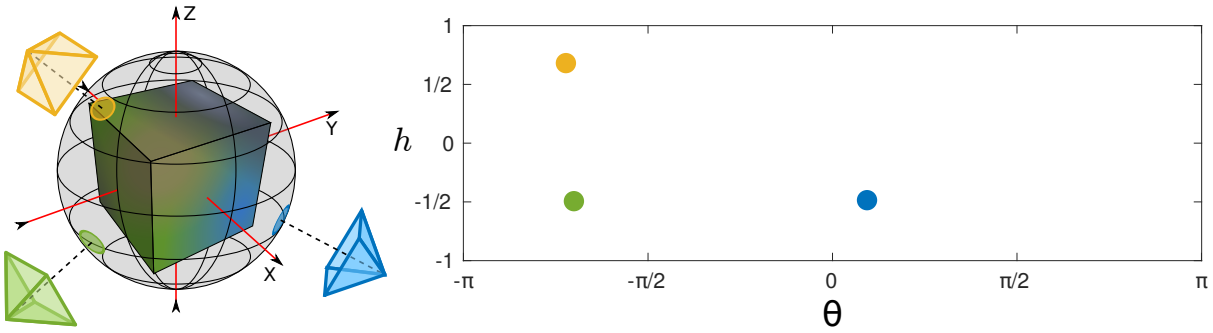


Figure 5.5: Cameras viewing the point (voxel) from different directions are mapped onto the θh plane.

The direction of the camera, *i.e.*, a point over the sphere, can be described in spherical coordinates by the angles $-\pi \leq \theta \leq \pi$ (azimuth or longitude) and $\pi/2 \leq \varphi \leq \pi/2$ (polar or latitude). It is often more convenient to represent the sphere surface using a cylindrical equal-area projection, replacing φ with $h = \sin(\varphi)$. We refer to this as the θh plane, which is illustrated in Figure 5.5.

The best sampling of the plenoptic function is acquired when cameras are uniformly distributed around the sphere since, in this way, it can equally capture light emanating from every direction. Let us assume a uniformly distributed probability density function of cameras around the sphere. In spherical coordinates, the probability of a camera being in the region delimited by $\phi_0 \leq \phi \leq \phi_1$ and $\theta_0 \leq \theta \leq \theta_1$ is given by

$$\int_{\phi_0}^{\phi_1} \int_{\theta_0}^{\theta_1} \frac{dA}{4\pi} = \int_{\phi_0}^{\phi_1} \int_{\theta_0}^{\theta_1} \frac{cos(\phi)}{4\pi} d\theta d\phi, \tag{5.2}$$

where 4π is the total surface area of an sphere with unitary radius and dA is the differencial area element in spherical coordinates. dA was replaced by its equivalente in terms of $d\phi$ and $d\theta$ (see Figure 5.6).

To change from spherical coordinate to the θh plane, we need to perform the following change of variable

$$h = sin(\phi)$$
$$dh = cos(\phi) d\phi.$$

Applying this modification in equation (5.2), we get

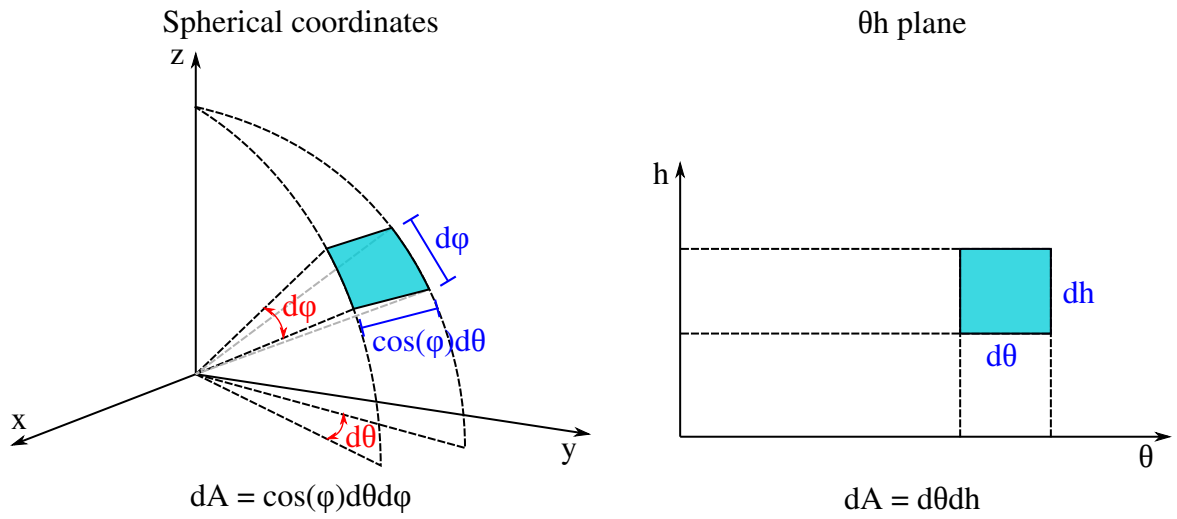


Figure 5.6: Differential area elements in spherical coordinates, assuming that the radius is equal to one, and in the θh plane.

$$\int_{sin(\phi_0)}^{sin(\phi_1)} \int_{\theta_0}^{\theta_1} \frac{1}{4\pi} d\theta dh = \int_{h_0}^{h_1} \int_{\theta_0}^{\theta_1} \frac{1}{4\pi} d\theta dh. \tag{5.3}$$

Noting that $d\theta dh$ is the differential area element of the θh plane, we conclude that a uniformly distributed probability density function in spherical coordinates results in a uniformly distributed probability density function in the θh plane and vice-versa. Therefore, it should be appreciated that a better sampling of the plenoptic function may be obtained if the camera views are uniformly spread over the θh plane.

We can generate a two-dimensional map over the θh plane with a projection of the colors as seen in every direction captured by the cameras (Figure 5.5). The next step is to apply a 2D transform to this map, but we need to cope with its sparsity. For so, one may divide the camera directions (θh) plane into sub-regions of the same area, as depicted in Figure 5.7. Each area is a quantized description of the camera direction. We may further divide each sub-region several times until attaining the desired precision. The smaller the sub-region, the more precise the camera position is represented. The number of divisions should be chosen to make the position information for two or more cameras not to fall within the same sub-region. In Figure 5.7, after dividing the plane, several sub-regions remain unoccupied. This representation is similar to voxelized point clouds in the 3D space. Therefore, we apply RAHT [29] to the colors associated with each camera (sub-region), through a 2D quad-tree decomposition rather than the 3D octree.

The hierarchical transform generates $N_c - 1$ high-pass coefficients and one DC coefficient ordered according to the depth of the quad-tree where they were generated. Deeper coefficients are associated with higher frequencies.

We combine the RAHT over the θh plane with the RAHT over the voxel spatial coordinates (xyz) in two ways:

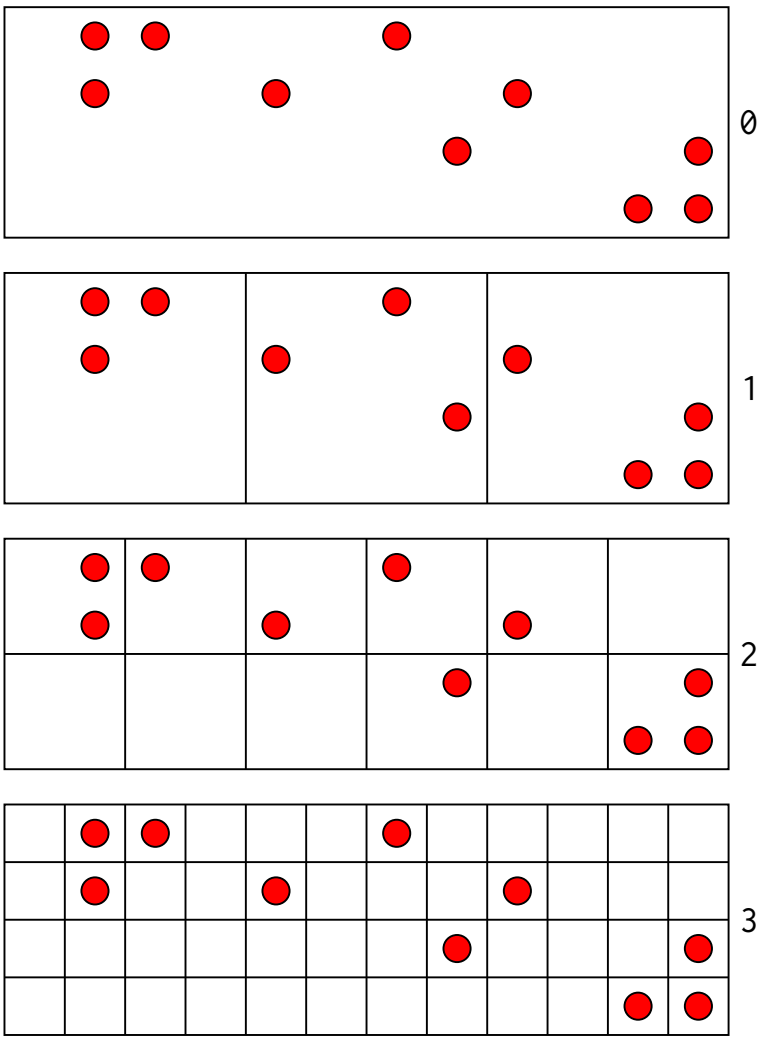


Figure 5.7: The color captured by each camera is projected onto the θh plane according to the direction where they see the voxel surface. The plane is then divided into sub-regions. The number of divisions defines the precision in which the camera's position is represented. The final subdivision is associated with a color, or it is empty if there are no cameras in that position.

- **RAHT-1:** In RAHT, we start from the leaves and transform attributes all the way to the root (DC value). Once the DC value of the cameras is obtained, we propagate it into the voxel space (xyz coordinates), i.e., the DC value becomes the color of the voxel, and we follow the rest of RAHT until reaching an overall DC value for all voxels and cameras. The coefficients to be encoded, using the same entropy coding as in [29], are the AC values of the cameras (θh space), the spatial AC values (xyz space), and the DC value.

- **RAHT-2:** When transforming the color samples in the θh plane we end up with N_c coefficients for each voxel: one DC value and $N_c - 1$ AC ones. Unlike the previously described approach, we assume all coefficients to be attributes, and all are subject to further processing for encoding. One can view it as if there were N_c point clouds, where voxel colors are the coefficient values, all sharing the same geometry. Each cloud is then transformed and encoded using the RAHT coder.

5.1.2 1-dimensional transform of the color vector

Let the colors of the n-th voxel be $R_i(n)$, $G_i(n)$ and $B_i(n)$, for $1 \leq i \leq N_c$. The colors are transformed into YUV space and let C represent one of the color components such that

$$\mathbf{c}(n) = [C_1(n), C_2(n), \ldots, C_{N_c}(n)]^T \tag{5.4}$$

represents that color component for a given voxel. If there is a linear transform $\mathbf{H}$ such that

$$\mathbf{s}(n) = [S_1(n), S_2(n), \ldots, S_{N_c}(n)]^T = \mathbf{H}\,\mathbf{c}(n)\ , \tag{5.5}$$

then each transformed signal $S(n)$ can be viewed as an attribute of a point cloud to be transformed and encoded. For a trichromatic color space such as YUV, there would be $3N_c$ data sets (point clouds) to undergo the RAHT transforming and encoding procedures, all sharing the same geometry.

In this approach, our preferred transform is the Kahunen-Loève transform (KLT). Briefly, we first compute the mean of the C samples, per camera, per color component, i.e.

$$\mu_C^i = \frac{1}{N_C} \sum_{n=1}^{{}^C N} C_i(n). \tag{5.6}$$

We, then, remove the mean and compute the $N_c \times N_c$ covariance matrix $\mathbf{\Gamma} = \{\Gamma(i,j), 1 \leq i,j \leq N_c\}$ among the camera signals $\{C_\ell(n)\}$. Hence,

$$\Gamma(i,j) = \frac{1}{N_C - 1} \sum_{n=1}^{N_C} \left(C_i(n) - \mu_C^i\right)\left(C_j(n) - \mu_C^j\right). \tag{5.7}$$

We compute eigenvectors and eigenvalues of $\mathbf{\Gamma}$ through the singular value decomposition as $\mathbf{\Gamma} = \mathbf{H}\mathbf{\Lambda}\mathbf{H}^T$, where $\mathbf{H}$ contains the eigenvectors of $\mathbf{\Gamma}$ as its columns and $\mathbf{\Lambda}$ is a diagonal matrix with the eigenvalues of $\mathbf{\Gamma}$. $\mathbf{H}$ is the KLT of the camera vector signal and is used to transform each $\mathbf{c}(n)$.

As side information, it is necessary to send the covariance matrix $\mathbf{\Gamma}$ to the decoder, so that it can adequately calculate its inverse. Such information is sent in the file header using single precision (32 bits) floating-point numbers. As the covariance matrix is symmetrical, it is only necessary to send $N_c(N_c+1)/2$ elements for each color component, instead of N_c^2. We refer to this method as RAHT-KLT.

Its clear disadvantage is the computation of statistics and matrix inversions. A discrete cosine transform (DCT) has been successfully used to replace the KLT in regular 2D image compression. However, in 3D, there is no predefined concept of sequencing among the cameras, and there are many possible permutations of the cameras in building vector $\mathbf{c}$. It is desirable that the amplitudes may not abruptly vary from one sample to its neighbor. This may be a problem if the cameras

are randomly ordered. Therefore, for every voxel, its plenoptic appearance attributes are ordered according to the direction of the camera relative to the voxel, in a spiral manner, starting from the bottom, as depicted in Figure 5.8. Hence, we can select $\mathbf{H}$ as the $N_c \times N_c$ DCT, and we refer to this method as RAHT-DCT.

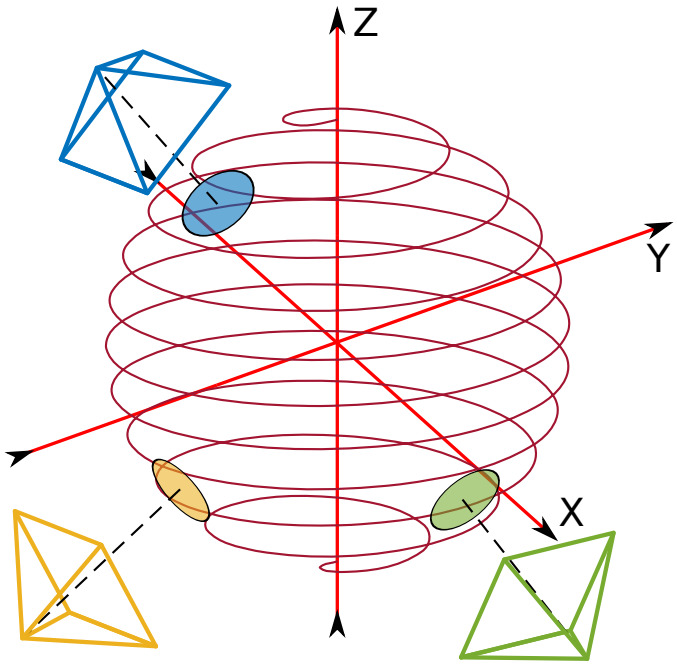


Figure 5.8: Camera ordering for the RAHT-DCT approach.

5.2 GEOMETRY MODIFICATION

Another alternative to encoding the plenoptic information is by subdividing voxels of the point cloud. Consider the voxel in Figure 5.9, whose colors are captured by five cameras placed on the depicted directions.

The sampled plenoptic information comprises not only the color but also the direction of the cameras. If we divide the voxel into M partitions along each axis ($M = 4$ in Figure 5.10), we obtain M^3 cubes with $1/M$ of the original width. Each of these cubes resulting from the division resembles voxels, and we refer to them as subvoxels.

We can use these subvoxels to introduce the plenoptic information if we associate each subvoxel with a camera. The position of the subvoxel represents the position of the camera. In order to associate the viewing direction and color to the subvoxel position, we devised two different methods. For the first one, named "face crossing point", the line connecting the voxel center and a camera, referred as viewing line, can be used to represent the viewing direction. This line crosses one of the subvoxels at the voxel's faces. Hence, in this method, the direction is represented by indicating the subvoxel position on the voxel's faces crossed by the given viewing line. The color as viewed in that direction is associated with this subvoxel (see Figure 5.10-(a)).

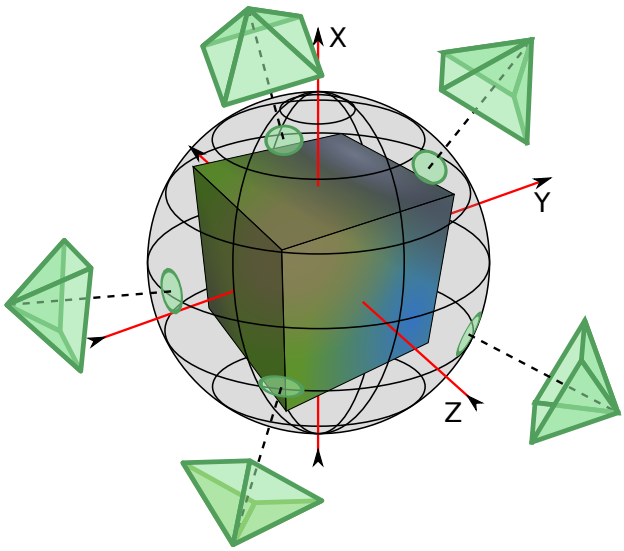
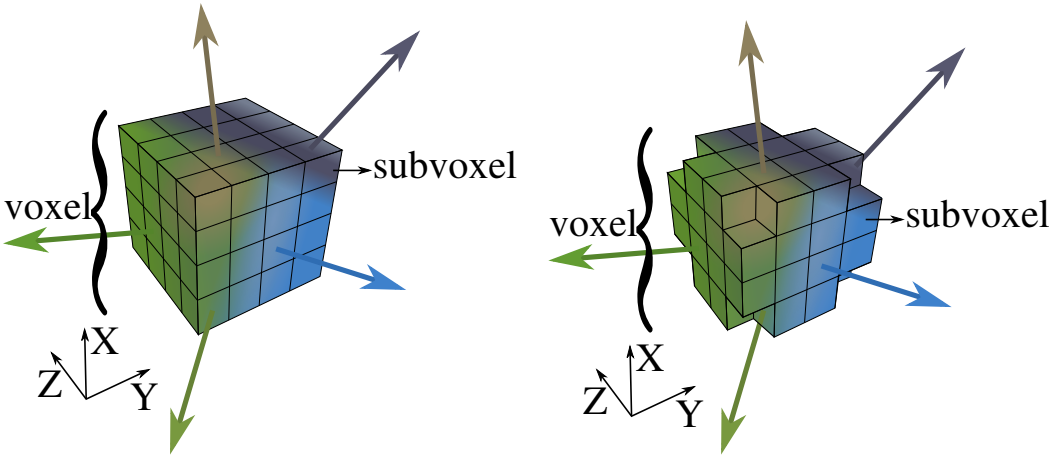


Figure 5.9: Capture of the plenoptic information of a voxel.



(a) Face crossing point

(b) Sphere crossing point

Figure 5.10: The figure depicts a voxel is division into subvoxels and how its subvoxels are employed to incorporated the plenoptic information.

All subvoxels that were not crossed by any viewing line remain unoccupied, as well as all the subvoxels not belonging to any of the voxel's faces.

We can improve the face crossing point method by, instead of using the subvoxels on the voxel's face, using those that are crossed by a sphere surface tangent to the voxel's faces. In this fashion, the occupied subvoxels are distributed in a spherical-like way, instead of a cube-like way, thus avoiding the distortions near the voxel's corner (see Figure 5.10-(b)). This method is named "sphere crossing point".

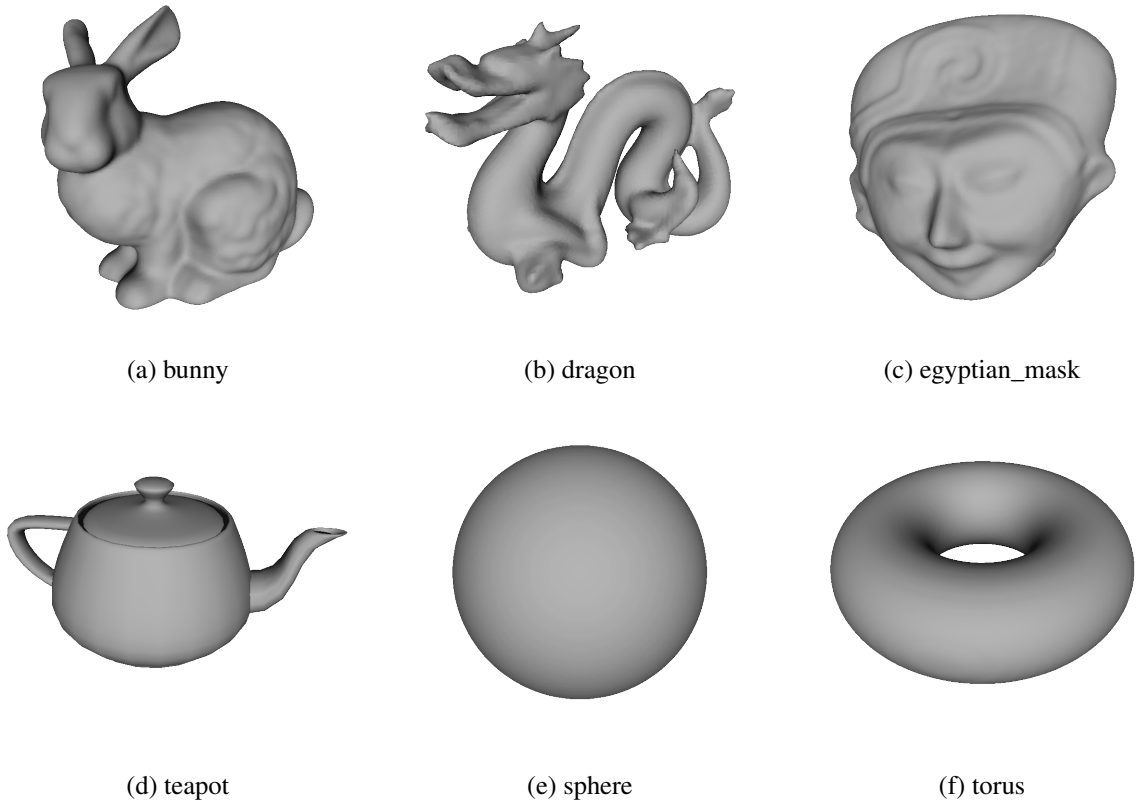
After attributing the plenoptic information to the subvoxels, we can now apply the RAHT-based coder [29] to the cloud of subvoxels. This process is transparent for RAHT because it treats the subvoxels as voxels. In this way, these two methods produce an expanded point cloud where each voxel was divided into N_c new voxels, were N_c is the number of cameras. The positions in

this expanded point cloud depict not only the position of the original voxels but also the direction associated with the viewing angle of the given color. The difference is that the expanded point cloud has only one color associated with it. In this work, we used RAHT to encode the expanded point cloud. However, any other method adapted to encode single-color point clouds can be applied as well.

The decoder needs the expanded geometry to be able to decode this plenoptic point cloud. Instead of conveying this expanded geometry directly with a geometry coder, we send camera positions to the decoder, and, mimicking the encoder, the decoder also expands the geometry to subvoxels.

5.3 SPECULARITY AND TRANSFORM PERFORMANCE

To ascertain the effectiveness of our algorithm for different object reflectivities, we carried tests with synthetic point clouds where we could control the specularity. The synthetic point clouds were created using widely known 3D models (as in Figure 5.11) obtained from the Stanford 3D Scanning Repository [95, 96] and the Utah teapot model created by Martin Newell in 1975 [97]. With 3D models we can easily adjust reflection characteristics.



(a) bunny (b) dragon (c) egyptian_mask

(d) teapot (e) sphere (f) torus

Figure 5.11: 3D Models from the Stanford 3D Scanning Repository.

We took 10 frames from 10 omnidirectional videos available at Youtube®. See Table 5.1.

The 3D models were voxelized using a depth of 9 and, thus, are contained in a box of size $512 \times 512 \times 512$ voxels. The θh plane was voxelized with a depth of 6 for the quad-tree. The omnidirectional images are projected within a sphere centered around the object, with a radius of 5×512. 20 cameras are uniformly distributed around the object at a radius of 2×512. Figure 5.12 is an out-of-scale representation of the setup used to synthesize the data.

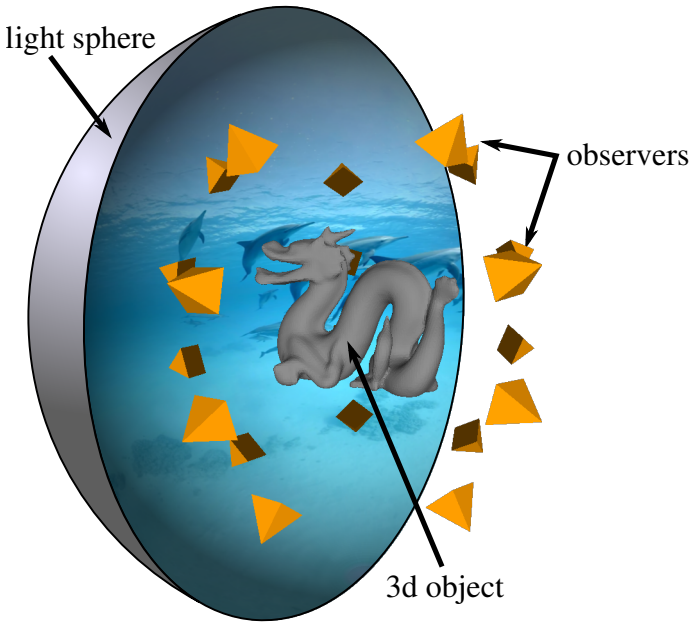


Figure 5.12: Synthetic data. A 3D object model is placed inside a sphere that acts as a light source. 20 obervers (cameras) are uniformly placed around the object.

Table 5.1: Spherical images.

#	**Video**
1	Wild Dolphins
2	Maldives
3	The Eye Of The Tiger
4	Underwater National Park
5	Fifty Shades Darker
6	NASA Encapsulation Launch of OSIRIS REx
7	Peru Presenta Cusco en Realidad Virtual
8	Clash of Clans Hog Rider
9	Angel Falls Venezuela
10	Hawaii The Pace of Formation

We adopted Phong's shading model [98] to compute the light reflected by each voxel in the directions observed by each of the 20 cameras. It has four parameters: the diffuse reflection constant k_d (ratio of light reflected in all directions); the specular reflection constant k_s (ratio of light emitted in the direction that a perfectly reflected ray would take) that we chose to be equal to $1 - k_d$; the ambient reflection constant that we chose to be zero; and the shininess constant,

which is more significant for surfaces that are smoother and more specular.

In order to evaluate the performance we fixed the bit rate at 0.2 bits/voxel/camera (in this work, we always mean bits per occupied voxel to describe bit-rates) and we computed the average peak signal-to-noise ratio (PSNR) between original and reconstructed attributes (colors), for a fixed set of reflection constants while varying the 3D model and spherical image. The diffuse reflection constant was varied between 0 and 1, while the shininess constant was assumed values of 5, 10, 50, and 800.

In objects that are more specular than diffuse, the light reflected from a voxel can highly vary according to the viewing angle, thus reducing correlation and the overall compression. This can be seen in Figure 5.13 where the PSNR of the decompressed image is higher for higher values of the diffuse reflection constant. The shininess constant also influences the performance as low values are associated with rough surfaces, while high values are associated with smooth ones. Rough surfaces tend to randomly deflect the direction of the reflected rays, producing more correlated appearance attributes. As shown in Figure 5.13, the higher the shininess constant, the lower the PSNR of the decompressed image for the same bit rate.

5.4 EXPERIMENTAL RESULTS

Figure 5.15 presents the rate-distortion curves comparing the methods sphere crossing point, face crossing point, and RAHT-1. Colors are represented in RGB space. In our experiment, the quantization step was varied between 15 and 500. A fourth rate-distortion curve referred as "independent" corresponds to apply RAHT to each camera color independently, *i.e.*, if a point cloud has available the color information of 13 cameras, for example, RAHT is applied 13 times, one for each camera information. This method was introduced just for the sake of comparison.

From the figure, it is clear that all these three methods have almost identical performance. This can be more clearly seen in Figure 5.16, which shows the PSNR difference between the methods when fixing the rate to the values shown in the abscissa. In Figure 5.16-(a) and (b), we observe that both the face and sphere crossing point methods outperform RAHT-1, and the difference is higher for higher rate values. Nevertheless, their difference is always below 0.5 dB in the range shown and can be considered negligible. The difference between face and sphere crossing point methods curves are even smaller, as shown in Figure 5.16-(c). All these three methods were able to outperform RAHT applied independently to each camera.

Figure 5.14 compares the performance among the methods for a fixed bit-rate (0.2 bits/voxel/-camera), but varying the appearance constants. To facilitate the comparison, the PSNR numbers are given relative to the performance of the RAHT-KLT method. When the diffuse reflection constant is below 0.6, RAHT-2 has a performance close to RAHT-KLT, presenting a distortion at most $0.5dB$ below that of RAHT-KLT. However, the performance of RAHT-2 quickly worsens compared with RAHT-KLT as k_d approaches 1.

Table 5.2: Bjontegaard average PSNR difference metric comparing RAHT-KLT, RAHT-DCT, RAHT-2 and RAHT-1 against RAHT-independent for all 6 point clouds

Point Cloud		**RAHT-KLT**	**RAHT-DCT**	**RAHT-2**	**RAHT-1**
Boxer	Y	**7.1085**	6.1475	5.6419	-2.1387
	U	**3.8839**	3.1761	2.8171	-0.6988
	V	**3.7828**	3.0820	2.7132	-0.7531
Longdress	Y	**16.9786**	16.5286	12.9858	10.4939
	U	**15.4572**	15.1915	11.4881	12.1208
	V	**15.7205**	15.4413	11.7386	12.4169
Loot	Y	**6.6654**	6.3731	5.1818	-0.1893
	U	**3.9890**	3.6152	2.8763	-0.4322
	V	**4.0598**	3.6526	2.9124	-0.3118
Redandblack	Y	**11.8950**	11.2875	8.9014	2.5070
	U	**10.0667**	9.7775	7.6094	4.7197
	V	**12.7393**	12.3447	10.0262	7.2953
Soldier	Y	**9.5781**	8.7311	7.6572	3.0744
	U	**5.8169**	5.3608	4.2784	1.7548
	V	**5.4982**	5.0869	4.0345	1.9260
Thaidancer	Y	**13.5818**	12.6944	10.3724	7.1546
	U	**12.3126**	11.7674	9.5354	7.0977
	V	**12.0782**	11.4855	9.2521	6.7789

Table 5.3: Bjontegaard percentage of bitrate saving metric comparing RAHT-KLT, RAHT-DCT, RAHT-2 and RAHT-1 against RAHT-independent for all 6 point clouds

Point Cloud		**RAHT-KLT**	**RAHT-DCT**	**RAHT-2**	**RAHT-1**
Boxer	Y	**-75.58%**	-70.91%	-66.74%	50.40%
	U	**-74.59%**	-70.20%	-63.67%	36.88%
	V	**-75.75%**	-71.41%	-64.81%	43.60%
Longdress	Y	**-89.92%**	-89.66%	-83.45%	-83.79%
	U	**-91.73%**	-91.60%	-84.44%	-89.98%
	V	**-92.14%**	-91.96%	-84.54%	-90.55%
Loot	Y	**-68.31%**	-65.10%	-57.96%	2.40%
	U	**-75.19%**	-72.59%	-64.29%	20.52%
	V	**-75.59%**	-72.91%	-64.68%	14.73%
Redandblack	Y	**-86.29%**	-85.28%	-78.24%	-34.67%
	U	**-89.28%**	-88.54%	-81.43%	-75.30%
	V	**-89.89%**	-89.40%	-82.03%	-79.53%
Soldier	Y	**-76.53%**	-73.67%	-68.79%	-35.38%
	U	**-82.67%**	-81.11%	-74.36%	-45.46%
	V	**-83.15%**	-81.71%	-74.91%	-52.22%
Thaidancer	Y	**-86.18%**	-84.57%	-77.53%	-68.13%
	U	**-89.77%**	-88.84%	-82.12%	-82.33%
	V	**-89.76%**	-88.86%	-81.91%	-81.57%

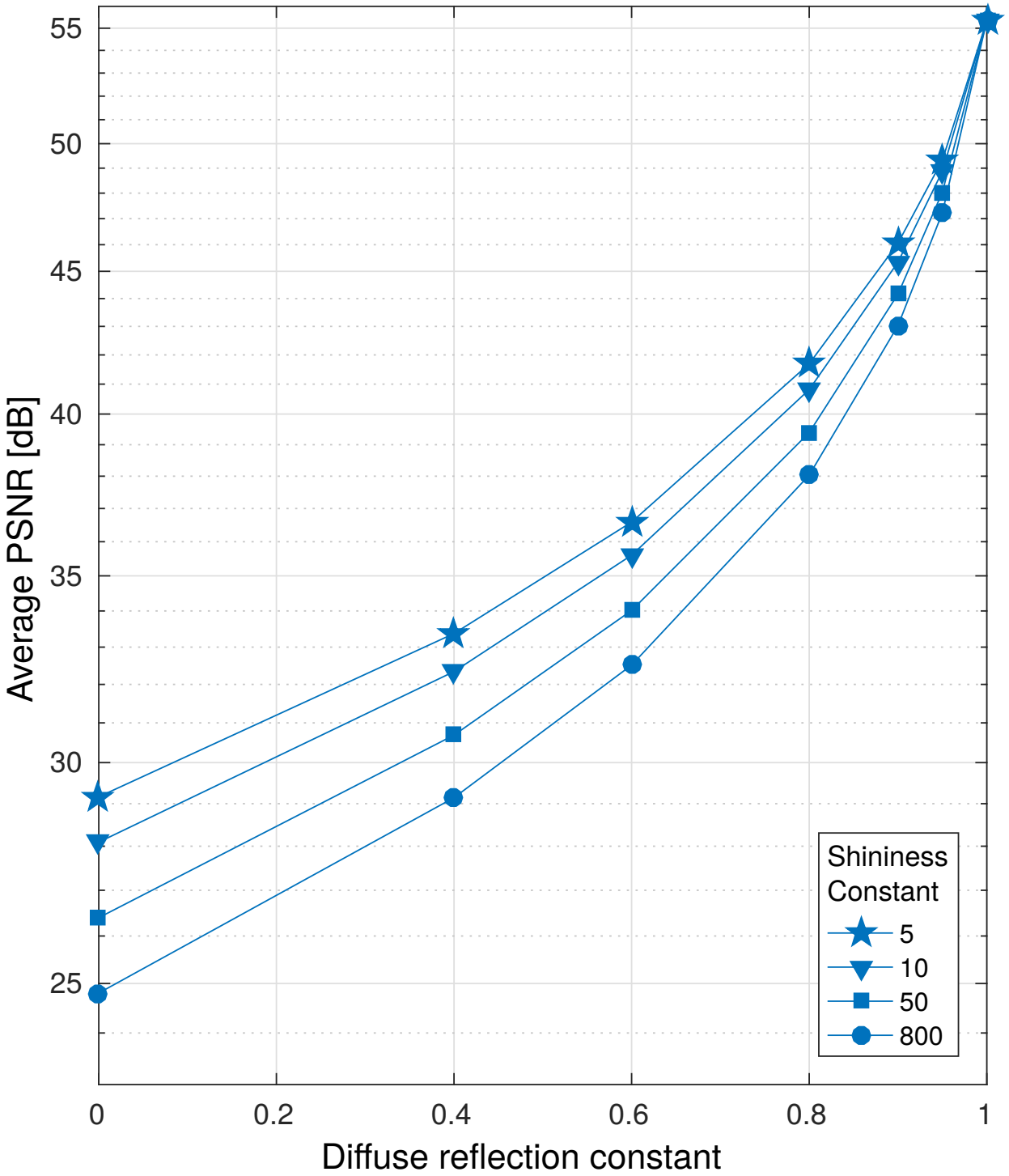


Figure 5.13: Distortion curves to evaluate the effect of diffuse reflection in the compression of the synthetic data. Bit-rate was set at 0.2 bits/voxel/camera, using the RAHT-KLT method.

We also carried tests on 5 realistic real-time-captured images. These images were voxelized using 11 bits of spatial resolution (octree with a depth level of $L = 11$), resulting in around 2 million voxels. The θh plane was created with a depth of 6 for the quad-tree. The colors are represented in the RGB space and transformed into YUV space for compression. The PSNR is evaluated only for the Y channel. In our experiment, the quantization step was exponentially

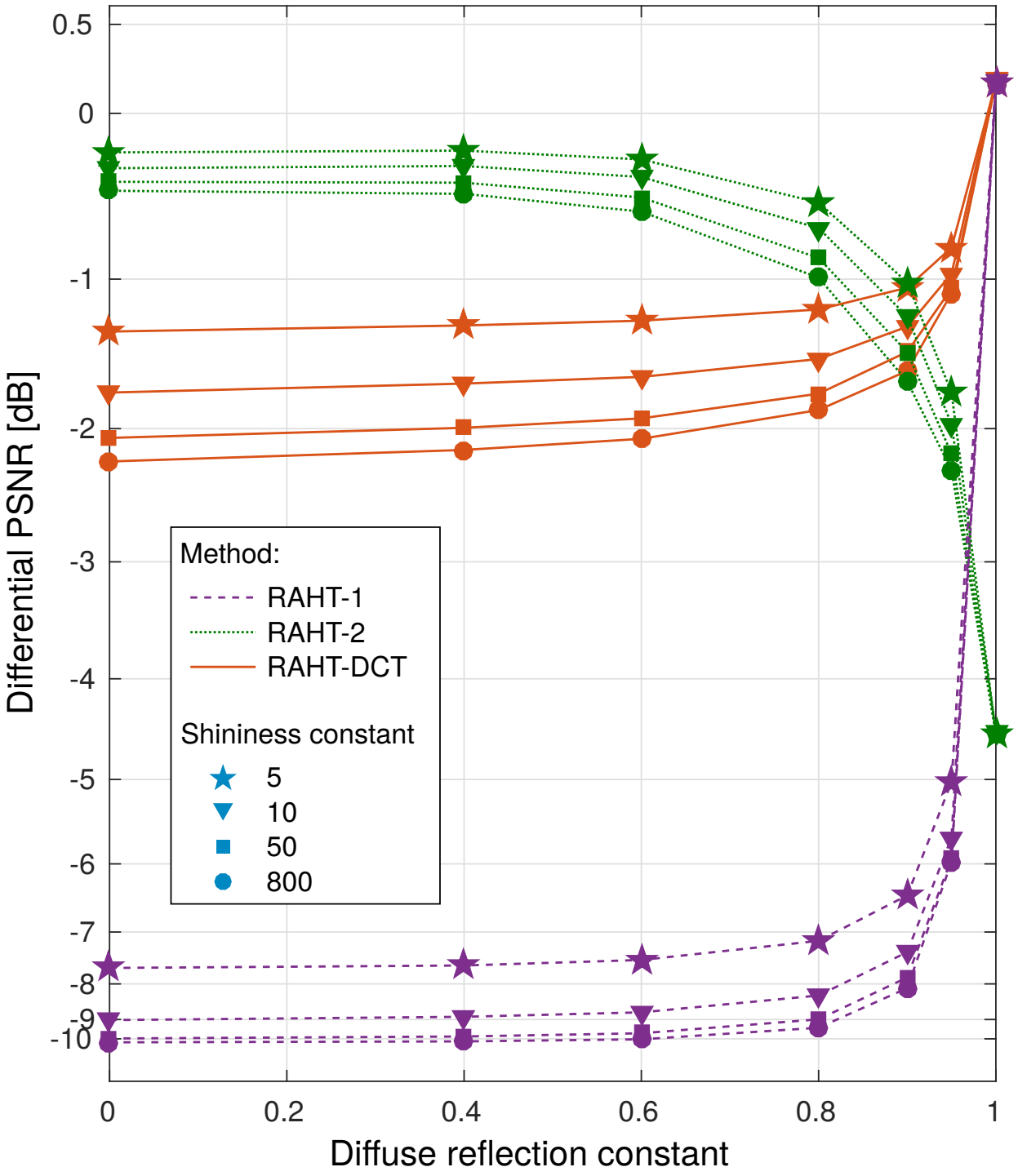
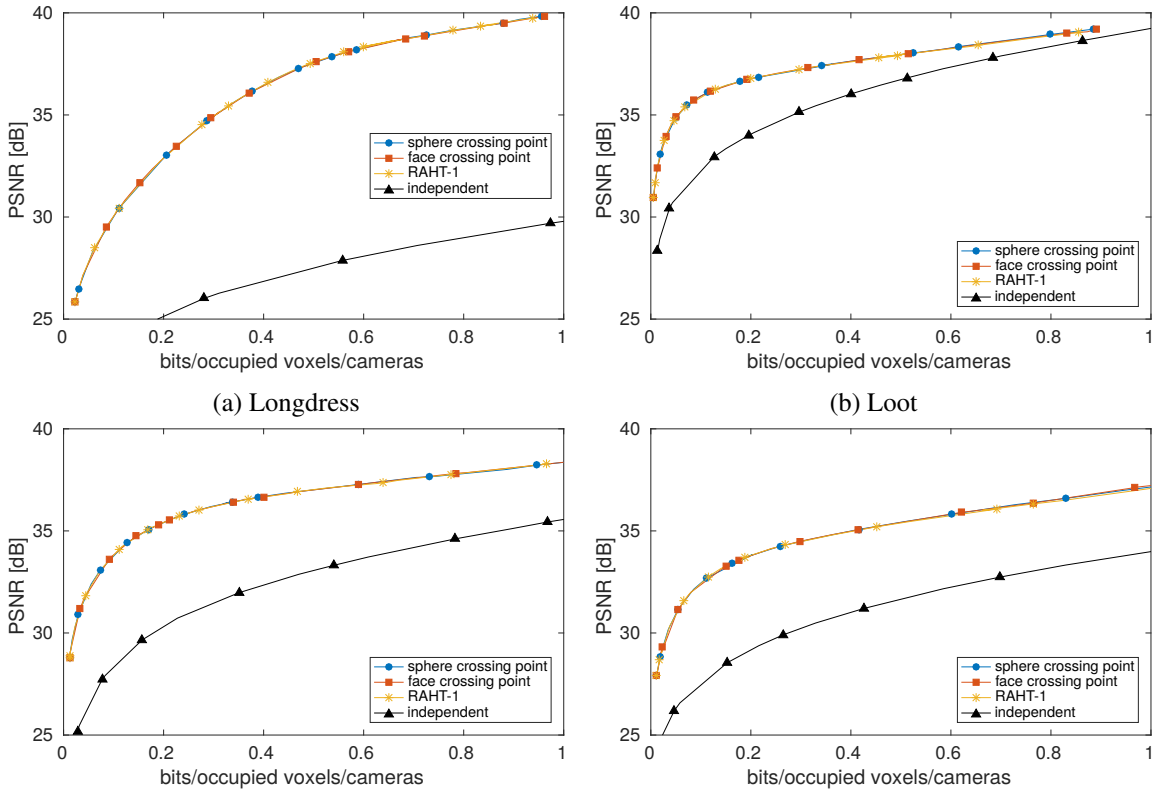


Figure 5.14: Performance comparison among the methods for the synthetic clouds in our test set, varying parameters. Data was compressed at 0.2 bits/voxel/camera, and PSNR was averaged over all 3D models and presented in differential form relative to the numbers of the RAHT-KLT method.

varied from 1.3 to 300. The results in terms of rate-distortion (RD) curves are shown in Figures 5.17–5.21.

The RAHT-KLT has been shown to provide the best performance among all methods. This is expected since its transform is adapted to the data statistics, encompassing all degrees of specu-



(a) Longdress (b) Loot

(c) Redandblack (d) Soldier

Figure 5.15: Rate-distortion curves comparing the methods sphere crossing point, face crossing point and RAHT-1 against RAHT applied independently to each camera color.

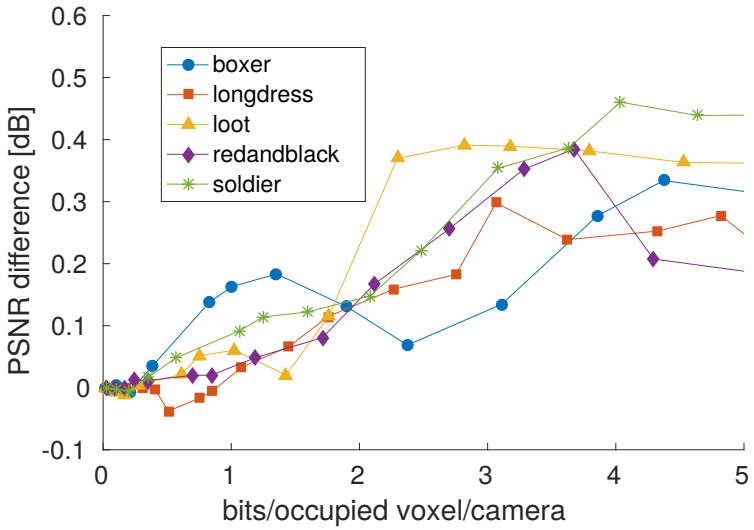
larity and diffuseness found for each point cloud.

The results using the real point clouds show that the methods can be ordered, from best to worst, as RAHT-KLT, RAHT-DCT, RAHT-2, and RAHT-1. With synthetic data, this is observed when $0.9 < k_d < 0.95$, which indicates that the objects in the real scenes are more diffuse than specular, but not Lambertian.

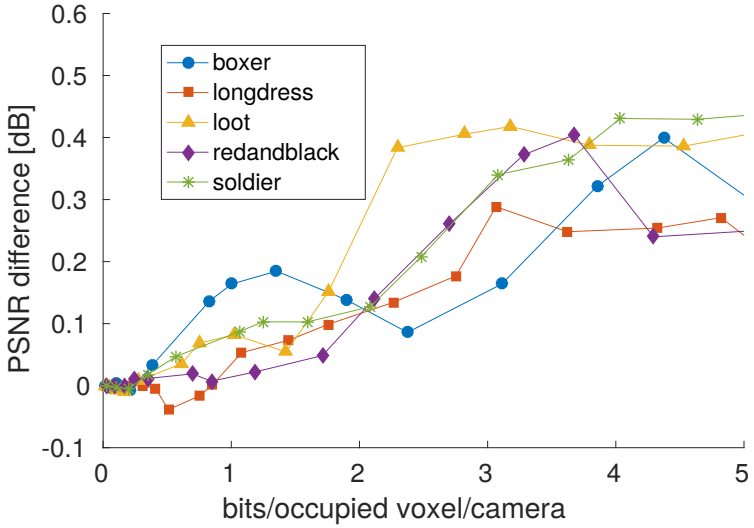
5.5 CONCLUSIONS

Objects with higher specularity may be less accurately represented by traditional single-color point clouds that are often used in real-time 3D capture and rendering. We extended the representation to encompass multiple-color voxels, which are samples of the plenoptic function for each voxel, thus referring to them as plenoptic point clouds. We have developed and tested four methods to extend the RAHT coder to compress plenoptic point clouds. RAHT-based coder is an excellent candidate for compression because of its simplicity of implementation allied with competitive rate-distortion performance. Two of the proposed approaches extend the RAHT over the sphere representing the plenoptic function (RAHT-1 and RAHT-2), while two other approaches involve the combination of RAHT with 1D transforms over the cameras color vector (RAHT-KLT

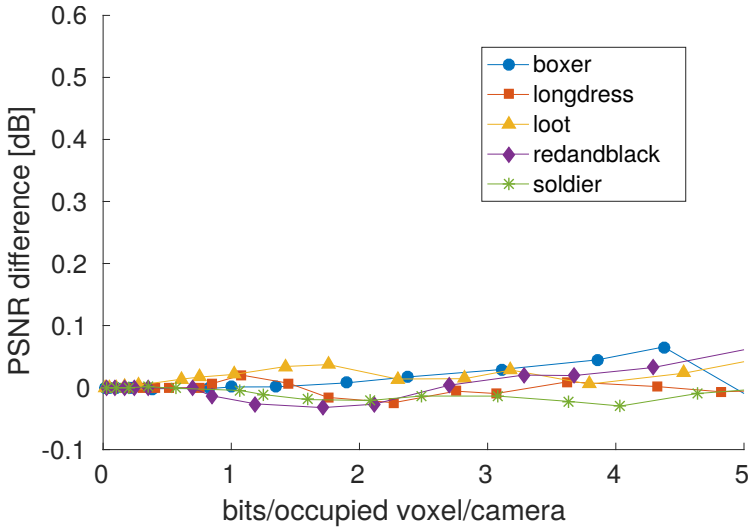
and RAHT-DCT). Among all four proposed methods, RAHT-KLT presents the best overall performance, even when the objects in the scene varies from completely specular to plainly diffuse.



(a) face crossing point - RAHT-1



(b) sphere crossing point - RAHT-1



(c) face crossing point - sphere crossing point

Figure 5.16: Difference of PSNR between methods for the same rate. We can see that the difference in performance among these three methods is very small. They have a virtually identical performance.

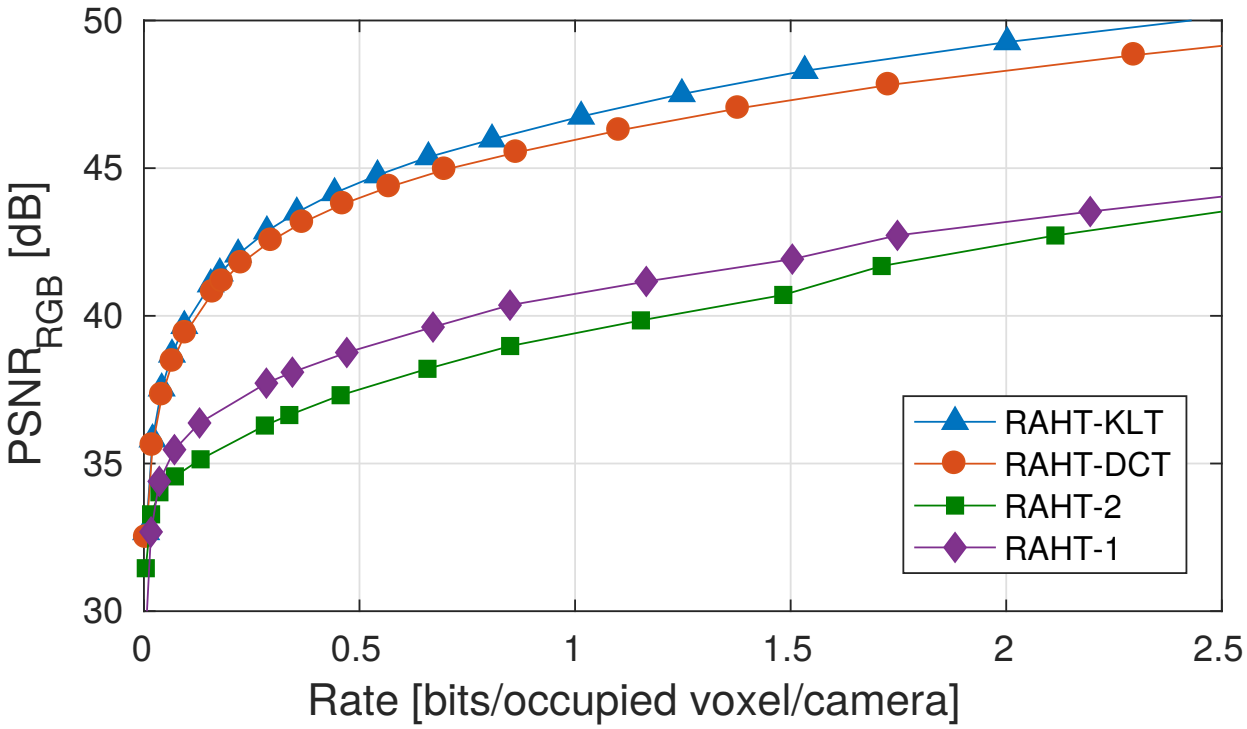


Figure 5.17: Rate-distortion curve for *Boxer*.

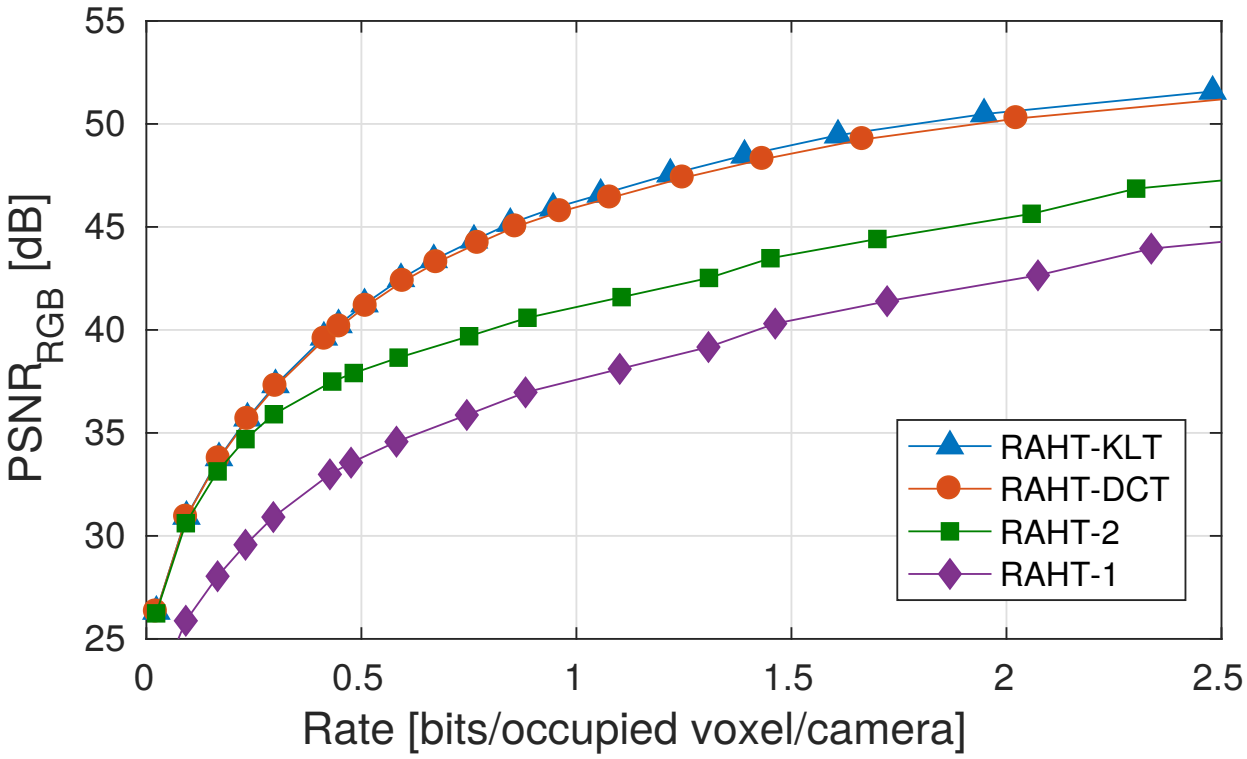


Figure 5.18: Rate-distortion curve for *Longdress*.

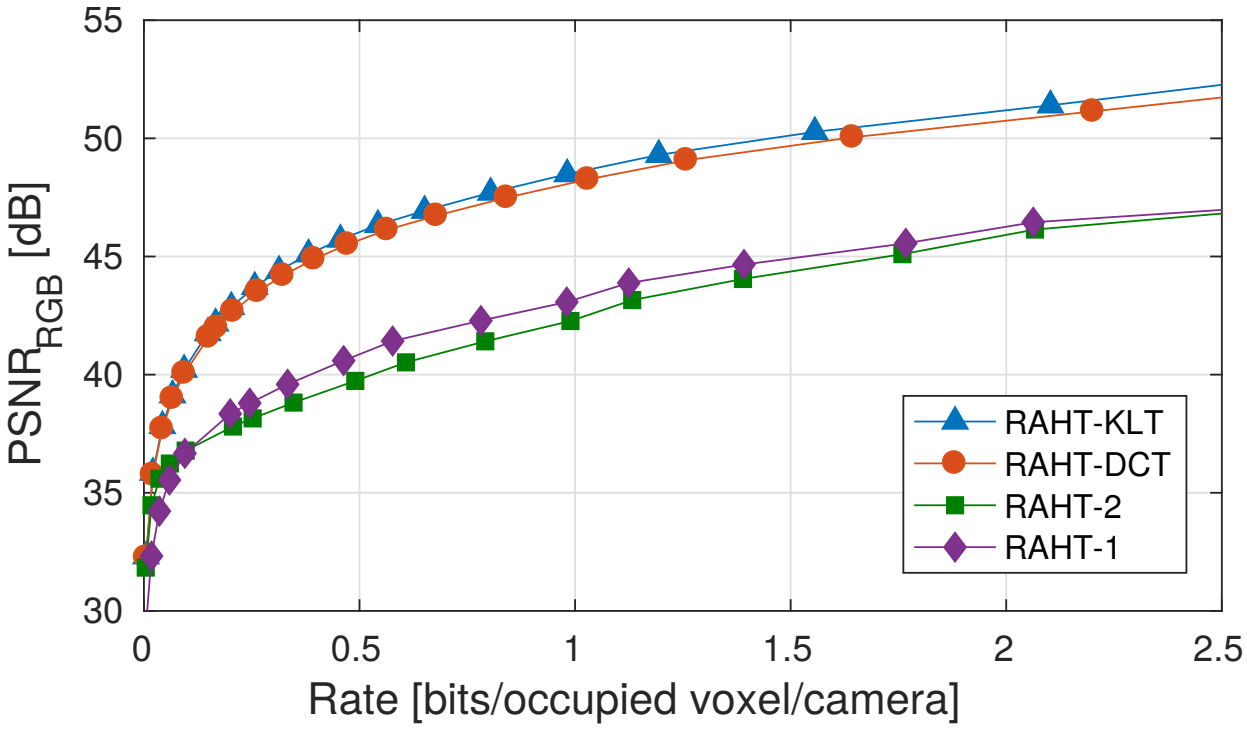


Figure 5.19: Rate-distortion curve for *Loot*.

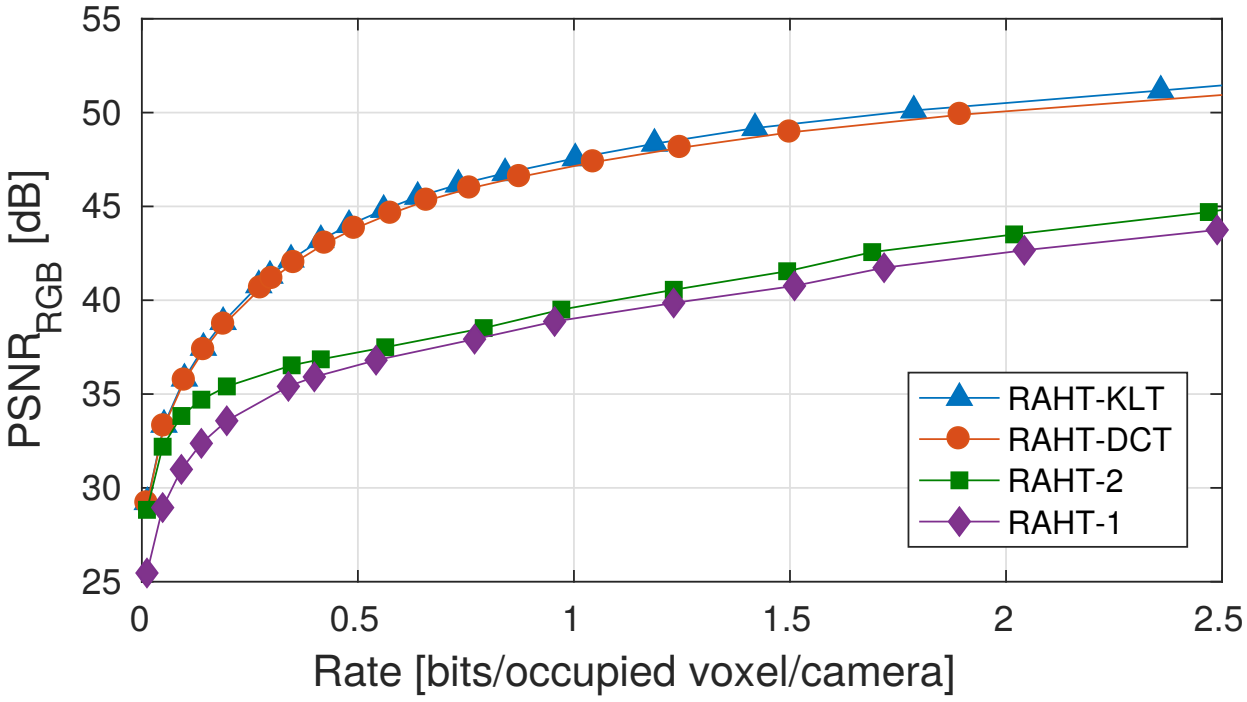


Figure 5.20: Rate-distortion curve for *Redandblack*.

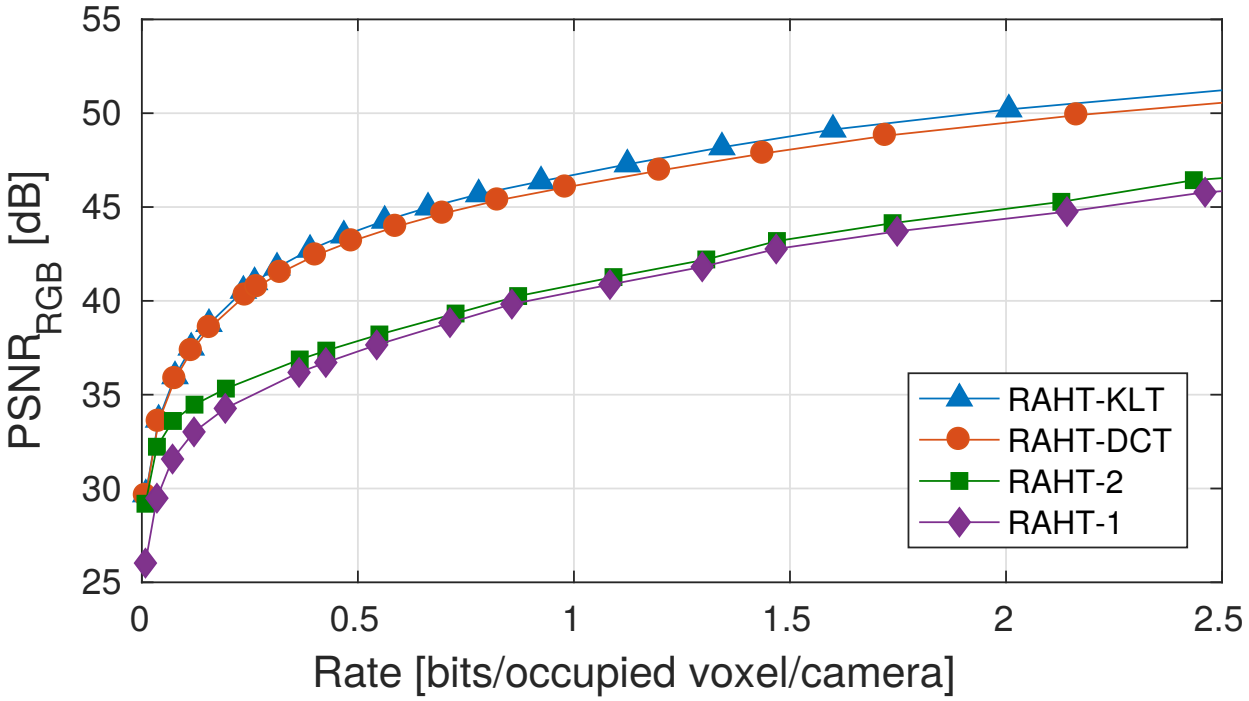


Figure 5.21: Rate-distortion curve for *Soldier*.

6 ENCODING OF REGIONS OF INTEREST

Human vision relies on two mechanisms. In the first, basic objects, like animals, faces, and plants, are categorized, and in the second, perception of the objects is enriched through conscious examination, such that the observer's attention is concentrated on regions of interest (ROI) that are relevant to the observer [99].

Like real scenes, point-cloud representations of scenes naturally also have ROI. As ROI attracts attention, preserving their quality is essential. For images and video, ROI-driven compression, or *ROI coding*, is well studied [100]. However, for point clouds, the literature on ROI coding is scarce.

In this chapter, we address the encoding of attributes with ROI. Inspired by a recent measure-theoretic interpretation of RAHT [101], in which RAHT is shown to be a separable 3D wavelet transform, that is orthonormal concerning a uniform counting measure on the set of points, we achieve ROI coding by modifying the measure, and then using RAHT as usual. Modifying the measure is equivalent to modifying the weights in a weighted-distortion measure. Hence, one may consider our approach to ROI coding as modifying the distortion measure following the ROI, and then coding to minimize the modified distortion measure, formally known as an input-weighted distortion measure [102–104].

The work present in this chapter was submitted as a paper to the IEEE Transactions on Image Processing and is still under review. Parts of this work was present in the International Conference on Image Processing [105] and at the XXXVII Brazilian Symposium on Telecommunications and Signal Processing [106].

Our approach to ROI coding has the advantage that it is codec-independent as we advocate using the ROI to modify the distortion measure. The modified distortion measure is then available to any codec for its rate-distortion optimization. There is a simple mapping from the ROI to the distortion measure, which can be quantified (for example, using perceptual experiments) independently of any particular codec. In our codec, we choose to transform coding with RAHT because RAHT is naturally optimized for the distortion measure by virtue of its measure-theoretic interpretation.

6.1 PROJECTION-BASED POINT CLOUD SEGMENT IDENTIFICATION

Many computer vision algorithms have been developed and are extensively studied for the 2D image case. Region identification is no exception. For point clouds, however, literature in the subject is scarce. We are interested in finding regions of interest, such as faces and hands in unstructured point clouds. In these, relative positions among voxels may have to be derived, and

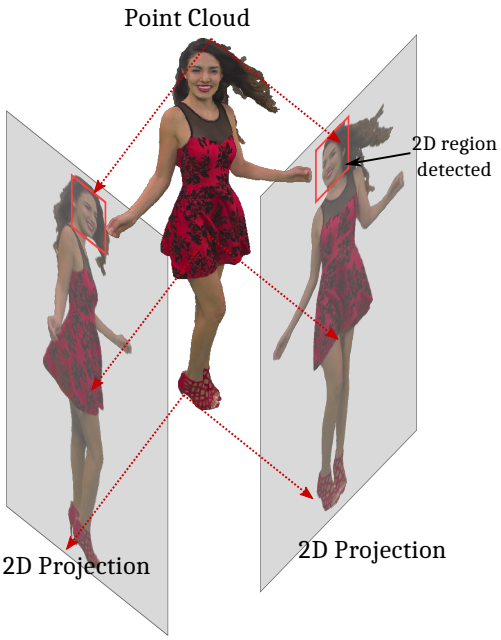


Figure 6.1: ROI detection in point clouds using projections and image identification algorithms. Detected pixels are mapped back onto voxels.

an efficient algorithm for feature identification is still being developed [107–109].

We recognize the level of difficulty in finding a solution, and we derived an alternative approach that proved itself useful. We do not know how to recognize features in a 3D point cloud space, but we know how to do it in 2D. Our solution is to use projections onto 2D images, to recognize the features in 2D space, and to map back the image pixels to the 3D voxels. In other words, computer vision tasks on 3D point clouds are performed on a 2D projection with the aid of a back-projection of 2D pixels onto 3D voxels. We fuse the data from many projections to find the voxels of interest to us. The idea is illustrated in Figure 6.1.

We begin by orthographically projecting the point cloud P onto a 2D plane I. If we imagine a voxel as a 3D cube and a pixel as a square element the size of the cube side, and if we orthogonally project the cube into any of its six faces, we may be able to uniquely map the voxel face to a pixel in the 2D projection plane. Hence, the $P \rightarrow I$ mapping would be reversible. If we project at any other oblique direction, the cube projection would not be square, but a more complex polygon. Such a projection would not fit into a square pixel and partially project onto many adjacent pixels. To cope with that situation, there are many solutions with varying degrees of accuracy and complexity. In $P \rightarrow I$ and $I \rightarrow P$, one solution is to compute the voxel or pixel color by linear combinations of the various partial projections. An alternative is to increase resolution by replicating voxels and pixels and simply assigning the voxel color to the pixel with the largest corresponding projection area. In the back-projection $I \rightarrow P$ we can mark the voxel whose center is the closest to the projection line from the center of a marked pixel in the 2D projection plane. After all voxels are marked, the point cloud should be reduced (downsampled or averaged) to the correct resolution. Similar interpolation issues arise if one does not assume

cubic voxels nor square pixels. Nevertheless, one should make sure we can map voxels to pixels and to map specific pixels back to voxels.

The projection-based region identification algorithm works as:

- Map the 3D voxels into a plane along the direction (θ, ϕ), where $-90^\circ \leq \theta \leq +90^\circ$ is the elevation and $0^\circ \leq \phi \leq 360^\circ$ is the azimuth.
- Identify the ROI in 2D, marking the pixels that belong to the ROI.
- Map the marked pixels back to the voxels in 3D. As a pixel may map to multiple voxels, one may use rounding or another decision process.

With the above algorithm, given a voxelized point cloud and a pair (θ, ϕ), we obtain a set of identified (marked) voxels. We, however, test many directions, since we do not know which orientation would be the best to identify the feature. We scan the (θ, ϕ) space by spanning θ from θ_{min} to θ_{max} in steps of $\Delta\theta$ and ϕ from ϕ_{min} to ϕ_{max} in steps of $\Delta\phi$. If we have N_θ elevation and N_ϕ azimuth points, we end with $N_a = N_\theta N_\phi$ projection and back-projection cycles. We do not always successfully identify an ROI. For example, only at a few viewpoints can we identify the faces of a person. Let us say that out of N_a projections we identify the ROI N_R times. Then for each of the N_R projection cycles, we record the voxels that are marked "of interest". If a given voxel is marked "of interest" N_i times, we then consider that particular voxel as belonging to the ROI if $N_i/N_R > \tau$, where τ is a given threshold. This information is fused by voting on voxels.

The algorithm is generic in nature, but we are focusing here as identifying faces of human point clouds as ROI. For that we use a traditional Viola-Jones algorithm [110] for face detection in images. This work is not about new face detection algorithms, and we used the most established algorithm to keep our focus on the projections and the compression. In the examples in this work, we used $\theta_{min} = -70^\circ$, $\theta_{max} = 90^\circ$, $\phi_{min} = 0^\circ$, $\phi_{max} = 359^\circ$, $\Delta\theta = \Delta\phi = 10^\circ$. Hence, $N_\theta = 17$ and $N_\phi = 36$ so that we perform $N_a = 612$ projections for each point cloud. Detection is made with $\tau = 0.3$, i.e., the voxel is assigned as a face voxel if it has been identified as such in at least 30% of the projections.

Face detection may be problematic when viewing the face sideways, which causes gaps in our face ROI in the cheeks and ears. To overcome this problem, we dilate the ROI mask in a particular way. Its Morton code can address each voxel in an integer grid. Morton codes are obtained by interleaving the binary representation of the voxel xyz integer coordinates, from the most to the least significant bit [111]. If the point cloud has depth d, the Morton code has $3d$ bits. The first $3(d-w)$ bits of the code is an address to cubes of $2^w \times 2^w \times 2^w$ voxels. Our dilation algorithm works with cubes of width 2^w, such that if a voxel in the cube is deemed in the ROI, all other occupied voxels in the cube are also made part of the ROI. These other voxels in the cube are easily found since they all share the same $3(d-w)$ initial bits of their Morton codes. In other words, if a voxel is deemed in the ROI, all other voxels with the same $3(d-w)$ bits prefix in their Morton codes are also made part of the ROI. Figure 6.2 shows results of expanding the ROI using different widths (2^w).

(a) *width = 1* (b) *width = 2* (c) *width = 4* (d) *width = 8*

Figure 6.2: Artifacts in the ROI identification caused by the obliquity of the projections and the Morton-code-based dilation for different cube widths.

In videos, temporal consistency is defined as the resemblance between one frame and the subsequent frame. This consistency is strongly explored in different ways such as to compress videos more efficiently [112], object tracking [113], disparity estimation [114], etc. In dynamic point clouds, we assume objects and regions do not change too much from one frame to another. For that, we adopt a speed-up to explore the temporal consistency among frames of a sequence. For a given frame, we calculate the preferred viewing direction, or a center mode (θ_c, ϕ_c), in the (θ, ϕ) space. We want to find the direction with the most identified voxels, assuming the projection region with the most pixels and back-projected voxels may correspond to a frontal unobstructed view of the face or of other features we are trying to identify. We take the weighted average (centroid) of the distribution of views. If at the i-th view direction at position (θ_i, ϕ_i) we identified n_i voxels then

$$\theta_c = \frac{\sum_i n_i \theta_i}{\sum_i n_i} \quad , \quad \phi_c = \frac{\sum_i n_i \phi_i}{\sum_i n_i}. \tag{6.1}$$

Assuming the center mode would not vary much from one frame to another, we concentrate our search for the next frame using new values of $\Delta\theta$, $\Delta\phi$, τ, while using $\theta_{min} = \theta_c - \theta_s$, $\theta_{max} = \theta_c + \theta_s$, $\phi_{min} = \phi_c - \phi_s$, $\phi_{max} = \phi_c + \phi_s$, where θ_S and ϕ_S define the search region. In our tests, the inter-frame mode uses $\theta_s = \phi_s = 20°$, $\Delta\theta = \Delta\phi = 4°$, and because of the more focused search space ($N_a = 121$) we raise the threshold to $\tau = 0.5$.

In summary, the algorithm is based on

1. For $\theta = \theta_{min} : \Delta\theta : \theta_{max}$
   For $\phi = \phi_{min} : \Delta\phi : \phi_{max}$
   -project the point cloud onto direction (θ, ϕ);
   -run 2D object detection algorithm (e.g. Viola-Jones);
   -back-project "marked" pixels as "marked" voxels;
   -count N_R;
   -count N_i for each voxel.

2. For each voxel, mark it as ROI if $N_i/N_R > \tau$.

3. Dilate ROI using blocks (e.g. 8-sided, or $w = 3$).

Note that with inter-frame consistency the (θ, ϕ) search space is narrowed around the previous-frame centroid (θ_c, ϕ_c), with smaller $\Delta\theta$, $\Delta\phi$ and larger τ.

6.2 ROI SIGNALING

The ROI location for a given frame needs to be conveyed to the decoder. Let $\mathbf{b} = \{b_i\}$ be a binary vector with values indicating whether occupied voxels belong to the ROI or not at that given frame. Voxels are sorted according to their associated Morton codes. Since neighboring voxels have a high probability of belonging to the same category (ROI or non-ROI) and the Morton codes tend to preserve neighborhood, we can convert vector $\{b_i\}$ in a differential vector $\{\overline{b_i}\}$ where $\overline{b_1} = b_1$ and $\overline{b_i} = 1$ if $b_{i-1} \neq b_i$ and 0 otherwise. Vector $\{\overline{b_i}\}$ is expected to have long sequences of zeros and is encoded with run-length coding. Since the vector is differential and binary, we only encode the runs of 0s. The run size is encoded with an adaptive Golomb-Rice code [61]. This is a simple coder; other, more sophisticated coders may be used as well. The number of bits spent signaling the ROI is typically not significant compared to the overall encoding bit rate.

6.3 ROI-WEIGHTED DISTORTION MEASURE

In lossy compression, artifacts introduced in some regions can profoundly influence the subjective perception of quality. The squared error may not correlate well with the subjective perception of quality.

Let us consider a generic attribute $\mathbf{X} = \{X_i\}$, where X_i is the attribute value associated with the i-th voxel of a total of N occupied voxels, and let $\hat{\mathbf{X}} = \{\hat{X}_i\}$ be its reconstruction. A better way to account for the relevance is to consider a *weighted squared error*, defined as

$$d(\mathbf{X}, \hat{\mathbf{X}}) = \sum_i w_i (X_i - \hat{X}_i)^2, \tag{6.2}$$

where w_i are the *weights* associated with each voxel and reflects its semantic or perceptual importance.

An ROI can be defined via the weights w_i. A natural choice is to set $w_i = 1$ for all voxels outside the ROI and $w_i > 1$ for voxels inside the ROI. A codec that minimizes this distortion measure subject to a rate constraint tends to reproduce X_i as $\hat{X}_i$ with squared error inversely proportional to w_i. It is also possible to define a smooth transition between the ROI border or define multiple ROIs with different weights.

The weights can be used to define a measure on the set of voxels composing the point cloud. A measure on a set is a systematic way to assign a number to each suitable subset of that set,

intuitively interpreted as its size. As each voxel has a weight associated with it, we define the measure of a subset S of voxels as

$$\mu(S) = \sum_{i \in S} w_i. \tag{6.3}$$

The measure is non-negative, as all weights are non-negative and the measure of two disjoint subsets is equal to the sum of their measure.

The definition of measure in Equation (6.3) induces the definition of the integral,

$$\int f(\mathbf{x}) d\mu(\mathbf{x}) = \sum_i w_i f(\mathbf{x_i}) \tag{6.4}$$

because

$$\frac{d\mu}{d\mathbf{x}} = \sum_i w_i \delta\left(|\mathbf{x} - \mathbf{x_i}|^2\right) \tag{6.5}$$

since voxels are dispersed in discrete positions $\mathbf{x_i}$.

The definition of the integral in Equation (6.4) in turn induces the definition of the inner product,

$$\langle f, g \rangle = \int f(\mathbf{x}) g(\mathbf{x}) d\mu(\mathbf{x}) = \sum_i w_i f(\mathbf{x_i}) g(\mathbf{x_i}), \tag{6.6}$$

which in turn induces the definitions of orthogonality, $f \perp g \Leftrightarrow \langle f, g \rangle = 0$, and norm, $||f|| = (\langle f, f \rangle)^{1/2}$. Altogether, these induce a Hilbert space. The weighted squared error given by (6.2) is precisely the squared norm $||f - \hat{f}||^2$ of this Hilbert space, where $f_i = X_i$ and $\hat{f}_i = \hat{X}_i$.

RAHT is region-adaptive to remain orthonormal regardless of the locations of the points. Recently RAHT has been shown to be interpretable as a separable piecewise constant spline wavelet that is orthonormal concerning the inner product $\langle f, g \rangle$ defined by the weights w_i [101]. Thus, if the weights are set to the weights in the ROI-weighted distortion measure, the transform remains orthonormal. Moreover, uniform scalar quantization of the transform coefficients with the quantization step size set to a constant minimize the ROI-weighted distortion measure, at least at high rates.

To be specific, let $\mathbb{R}^3$ be uniformly partitioned into cubes of size $2^{-m} \times 2^{-m} \times 2^{-m}$, half-cubes of size $2^{-m} \times 2^{-m} \times 2^{-(m+1)}$, and quarter-cubes of size $2^{-m} \times 2^{-(m+1)} \times 2^{-(m+1)}$, and let $\mathcal{F}_{3m}$, $\mathcal{F}_{3m+1}$, and $\mathcal{F}_{3m+2}$ be the spaces of all functions $f_\ell : \mathbb{R}^3 \to \mathbb{R}$ that are piecewise constant on these blocks, for $\ell = 3m$, $3m + 1$, and $3m + 2$, respectively. The nested sequence of function spaces $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \cdots \subseteq \mathcal{F}_\ell \subseteq \mathcal{F}_{\ell+1} \subseteq \cdots$ approximates ever more finely (with respect to the norm, i.e., the weighted squared error) the space of piecewise continuous functions.

Now let $B_{\ell,n}$ denote a block at level ℓ indexed by n, let $1_{B_{\ell,n}}(\mathbf{x})$ be its indicator function, and let $w_{\ell,n} = \mu(B_{\ell,n})$ be its measure. Then $\mathcal{F}_\ell$ is spanned by the basis functions

$$\phi_{\ell,n}(\mathbf{x}) = w_{\ell,n}^{-1/2} 1_{B_{\ell,n}(\mathbf{x})}, \tag{6.7}$$

which are orthogonal to each other and are normalized with respect to the inner product and norm induced by the weighted measure. Similarly, let $B_{\ell+1,n_0}$ and $B_{\ell+1,n_1}$ denote the sub-blocks of $B_{\ell,n}$, and let $\mathcal{G}_\ell$ be the orthogonal complement of $\mathcal{F}_\ell$ in $\mathcal{F}_{\ell+1}$. Then $\mathcal{G}_\ell$ is spanned by the basis functions

$$\psi_{\ell,n}(\mathbf{x}) = \frac{-w_{\ell+1,n_0}^{-1} 1_{B_{\ell+1,n_0}}(\mathbf{x}) + w_{\ell+1,n_1}^{-1} 1_{B_{\ell+1,n_1}}(\mathbf{x})}{(w_{\ell+1,n_0}^{-1} + w_{\ell+1,n_1}^{-1})^{-1/2}} \tag{6.8}$$

which are orthogonal to each other and to the functions given by Equation (6.7), and are normalized, as can be verified by the diligent reader. Thus any function $f_{\ell+1} \in \mathcal{F}_{\ell+1}$ can be written as

$$f_{\ell+1}(\mathbf{x}) = \sum_n F_{\ell,n}\phi_{\ell,n}(\mathbf{x}) + \sum_n G_{\ell,n}\psi_{\ell,n}(\mathbf{x}), \tag{6.9}$$

where the $F_{\ell,n} = \langle f_{\ell+1}, \phi_{\ell,n}\rangle$ are known as low-pass coefficients and the $G_{\ell,n} = \langle f_{\ell+1}, \psi_{\ell,n}\rangle$ are known as high-pass coefficients. After some algebraic manipulation, Equations (6.7) and (6.8) can be expressed recursively as the "two-scale equations"

$$\phi_{\ell,n}(\mathbf{x}) = a\phi_{\ell+1,n_0} + b\phi_{\ell+1,n_1} \tag{6.10}$$

$$\psi_{\ell,n}(\mathbf{x}) = -b\phi_{\ell+1,n_0} + a\phi_{\ell+1,n_1}, \tag{6.11}$$

where $a = \frac{\sqrt{w_{\ell+1,n_0}}}{\sqrt{w_{\ell,n}}}$ and $b = \frac{\sqrt{w_{\ell+1,n_1}}}{\sqrt{w_{\ell,n}}}$. Substituting these into the definitions of $F_{\ell,n}$ and $G_{\ell,n}$, we obtain

$$\begin{bmatrix} F_{\ell,n} \\ G_{\ell,n} \end{bmatrix} = \begin{bmatrix} a & b \\ -b & a \end{bmatrix} \begin{bmatrix} F_{\ell+1,n_0} \\ F_{\ell+1,n_1} \end{bmatrix}, \tag{6.12}$$

which is a Givens rotation whose angle of rotation depends on the relative weights of the sub-blocks.

RAHT applies Equation (6.12) recursively to expand $f_L \in \mathcal{F}_L$ as

$$f_L(\mathbf{x}) = \sum_n F_{0,n}\phi_{0,n}(\mathbf{x}) + \sum_{\ell=0}^{L-1}\sum_n G_{\ell,n}\psi_{\ell,n}(\mathbf{x}), \tag{6.13}$$

where L is chosen large enough so that each cube $B_{L,n}$ contains at most a single point, say $\mathbf{x}_i$ with value $f_i = f(\mathbf{x}_i)$. The number of coefficients is N, i.e., RAHT is critically sampled. (For details, see [101].) Note that $\phi_{L,n}(\mathbf{x}) = w_i^{-1/2} 1_{B_{L,n}}(\mathbf{x})$, and therefore $F_{L,n} = \langle f, \phi_{L,n}\rangle = w_i^{1/2} f_i$. This generalizes RAHT in [29], for which $w_i = 1$ for all points $i = 1, \ldots, N$.

The RAHT coefficients are uniformly scalar quantized with stepsizes $\Delta(F_{0,n})$ and $\Delta(G_{\ell,n})$, $\ell = 0, \ldots, L-1$, and are entropy coded. Because Givens rotations are orthonormal, norm is preserved. Thus the squared quantization error is

$$\sum_n (F_{0,n} - \hat{F}_{0,n})^2 + \sum_{\ell=0}^{L-1}\sum_n (G_{\ell,n} - \hat{G}_{\ell,n})^2 = \sum_{i=1}^{N} w_i (f_i - \hat{f}_i)^2, \tag{6.14}$$

which is the same as the ROI-weighted distortion measure. Since a constant stepsize $\Delta = Q_{step}$ minimizes the squared quantization error subject to an entropy constraint, at least at high rates

[102], setting the stepsizes of the RAHT coefficients to a constant also minimizes the ROI-weighted distortion measure desired for ROI coding.

In summary, with RAHT, at the encoder, voxels in ROI should have initial weights set to $w_i = w$ and initial attributes scaled by $\sqrt{w}$. The decoder should scale back the attributes.

6.4 EXPERIMENTAL RESULTS

We used 8 point clouds sequences to test our code: Longdress, Loot and Redandblack, voxelized with a depth of 10, and Andrew, David, Phil, Ricardo, and Sarah, voxelized with a depth of 9. The first 3 sequences have 300 frames while the last 5 have 318, 216, 245, 216 and 207 frames, respectively. Examples of projections of the identified faces for frames 1, 101, and 201 of each sequence are shown in Figures 6.3 and 6.4.

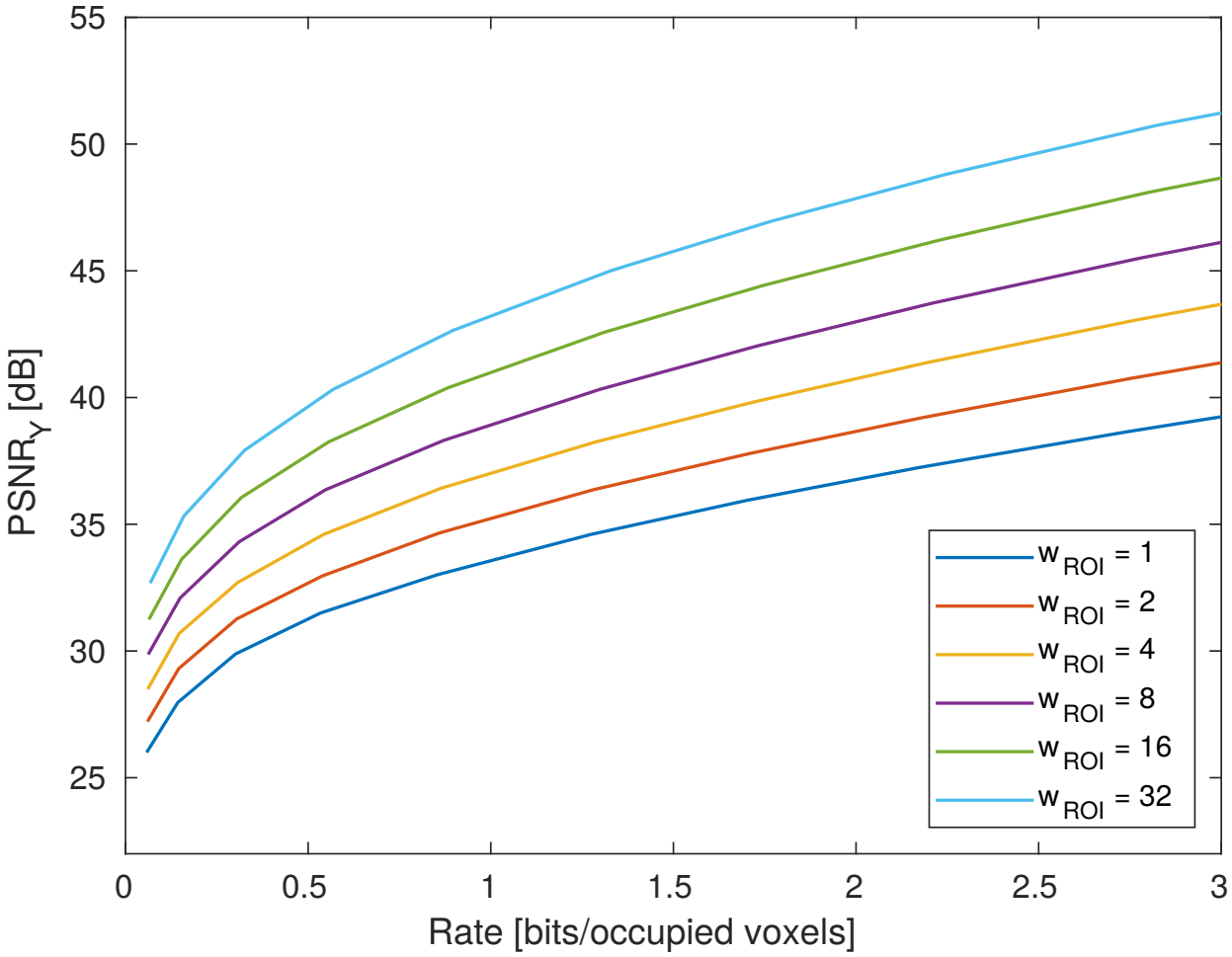
Figure 6.5 shows average rate-distortion curves for sequence Longdress. The bit-rate includes the overhead to signal the ROI. The weights for voxels in the ROI, w_{ROI}, were chosen as 1, 2, 4, 8, 16 and 32 while weights outside the ROI are 1. In Figure 6.5(a), the PSNR was computed for voxels inside the ROI against rate in bits per occupied voxels, while w_{ROI} is varied. As expected, when we increase w_{ROI}, the quality inside the ROI increases. When $w_{ROI} = 1$ all voxels have the same weight, and it is the regular encoder. In Figure 6.5(b), the PSNR was computed for voxels outside the ROI against rate in bits per occupied voxels, while w_{ROI} is varied. There is a minimal difference among curves, in this case, because the ROI region is typically much smaller than the rest. Re-allocating a very small bit-rate over a large area can largely increase the ROI bit-rate, thus improving the ROI while keeping the degradation to non-ROI to a minimum. As a result, the performance impact of increasing w_{ROI} is not very significant in Figure 6.5(b). Plots for the other sequences are qualitatively the same, and their inclusion would be repetitive and tedious.

Table 6.1 presents Bjontegaard-delta PSNR (BD-PSNR) and rate (BD-RATE) [60] for rates from 0.08 to 1.0 bit per occupied voxel (bpov). Table 6.1 reflects an average of results for all sequences and frames, uses all voxels as reference and compares against either the ROI or non-ROI voxels. When $w_{ROI} = 1$, the area computed for all voxels only slightly differs from the ROI and non-ROI, as expected. As the w_{ROI} increases, the area under the curve computed for voxels outside the ROI remains slightly inferior to the one computed for all voxels. The performance does not drop as much for voxels outside the ROI because the ROI is relatively small. Typically, less than 6% of occupied voxels are in the ROI. For voxels in the ROI, on the other hand, the performance significantly increases as w_{ROI} increases. If we look at the average difference between $PSNR_Y$ computed for voxels in the ROI and that computed for all voxels, we obtain values close to the expected, i.e values computed as $10 \log_{10}\left(\sqrt{w_{ROI}}\right)$ which are 1.5051, 3.0103, 4.5154, 6.0206 and 7.5257 dB when w_{ROI} is 2, 4, 8, 16 and 32, respectively.

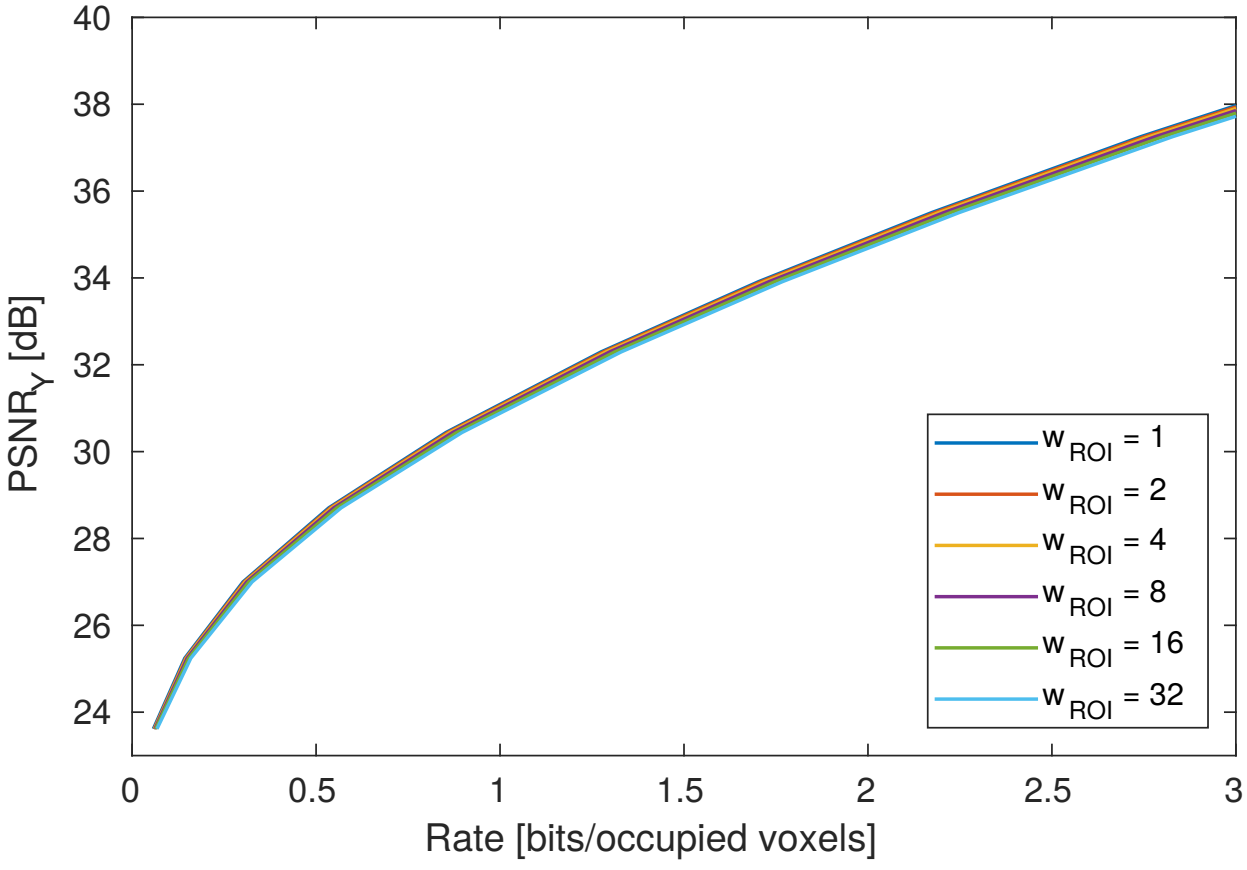
In the modification here introduced, the transform minimizes the weighted squared error given

Figure 6.3: ROI identified in frames 1, 101 and 201 for sequences Longdress, Loot, Redandblack and Phil.

Figure 6.4: ROI identified in frames 1, 101 and 201 for sequeces Ricardo, Andrew, Sarah, and David.



(a) ROI



(b) non-ROI

Figure 6.5: Average rate-distortion curves for all frames of the sequence Longdress. The PSNR is computed only for (a) voxels in the ROI and (b) voxels outside the ROI.

Table 6.1: Average BD-PSNR and BD-rate in the range from 0.08 and 1 bpov with the PSNR computed inside and outside the ROI, compared to the PSNR of all voxels.

	BD-PSNR		**BD-Rate**	
w_{ROI}	**ROI**	**non-ROI**	**ROI**	**non-ROI**
1	-0.26	0.33	23.46	-4.31
2	1.03	0.09	-11.93	0.77
4	2.47	-0.10	-36.68	4.98
8	3.97	-0.24	-53.53	8.52
16	5.53	-0.34	-64.88	11.48
32	7.16	-0.41	-70.18	14.01

by equation (6.2). The PSNR is computed as

$$PSNR = 10\ \log_{10}\left(\frac{255^2}{E_{noise}}\right), \tag{6.15}$$

i.e., the ratio between the peak signal and the noise energy E_{noise} that is computed as the average squared error. We can redefine energy as

$$E_{noise} = \frac{\sum_i w_i \left(Y_i - \overline{Y_i}\right)^2}{\sum_i w_i} \tag{6.16}$$

for the Y channel, for example to conform to Equation (6.2), which implies the definition of a weighted PSNR. Alternatively, an equivalent definition of the weighted PSNR is

$$PSNR = 10\ \log_{10}\left(\frac{E_{peak}}{\sum_i w_i \left(Y_i - \overline{Y_i}\right)^2}\right), \tag{6.17}$$

where $E_{peak} = \sum_i w_i 255^2$ is the peak signal energy under the weighting, which leaves the denominator as the distortion in Equation (6.2).

Figure 6.6 compares the average rate-distortion curves for the sequence Loot. When $w_{ROI} = 1$, the weighted and the standard PSNR are equal. However, when $w_{ROI} > 1$ the weighted PSNR shows better results than the standard PSNR for the same data. This is expected since our transform minimizes distortion given by Equation (6.2).

All results so far indicate a sizable improvement in the ROI at the expense of a small loss to the regions outside the ROI. The subjective analysis of quality is the primary motivation behind the use of ROI for compression. Figure 6.7 to 6.14 shows a frame of each sequence in two scenarios: (b)(d) where all voxels are equally encoded and (c)(e) where we privilege voxels in the ROI (face). Even though the point clouds in (b) have a higher overall PSNR, the point cloud in (c) has a better PSNR for voxels in the ROI and seems to have a much higher subjective quality since humans are susceptible to face artifacts. Some point clouds in Figure 6.7 to 6.14 (*e.g.*, the last three) have a not-so-large PSNR gain in the ROI against the non-ROI. This is the case where there are not many color details outside the ROI. Hence there are already fewer bits assigned to it. Hence, it is more challenging to shift bits from the non-ROI to the ROI voxels.

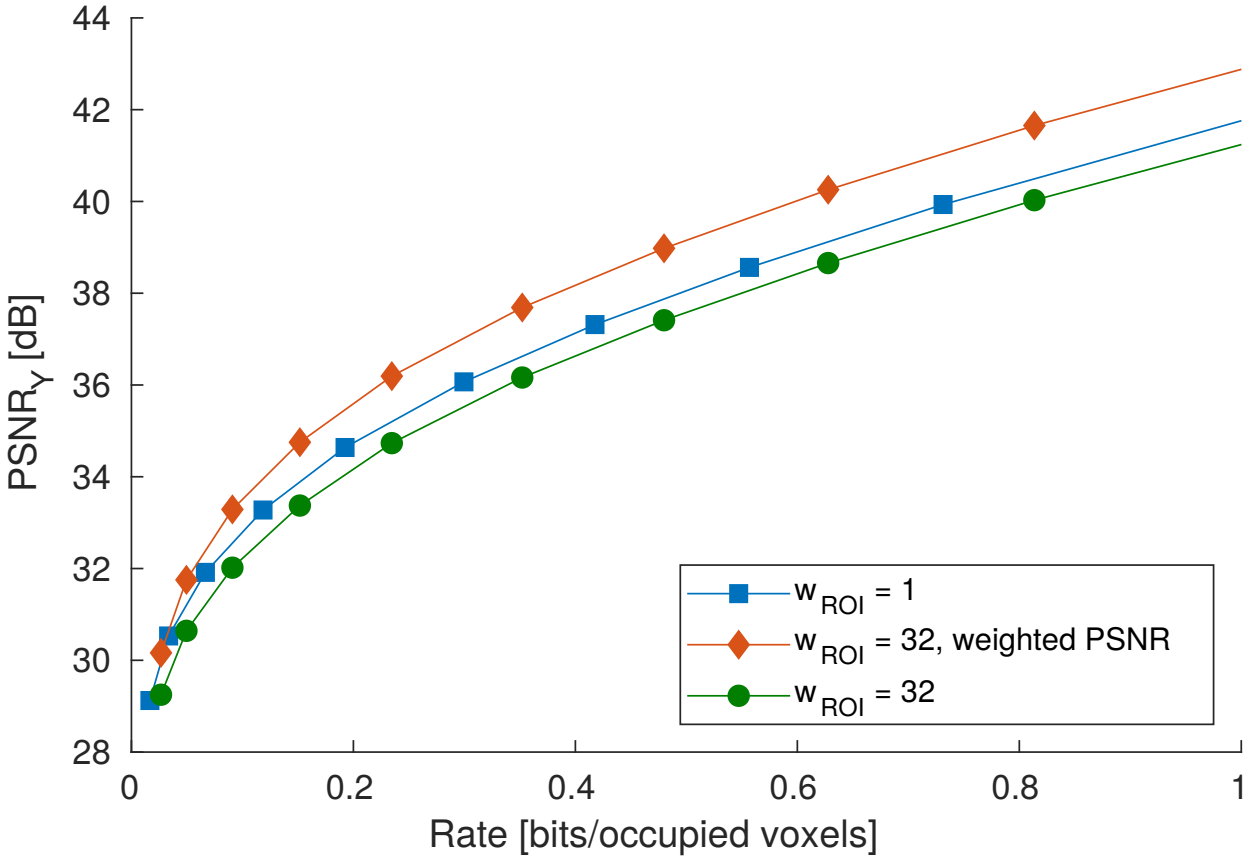
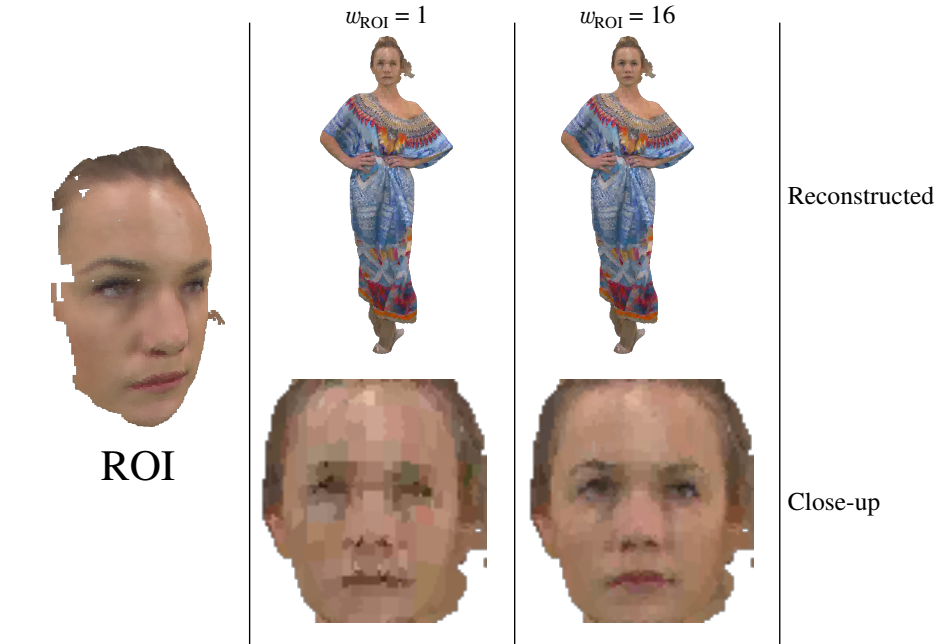
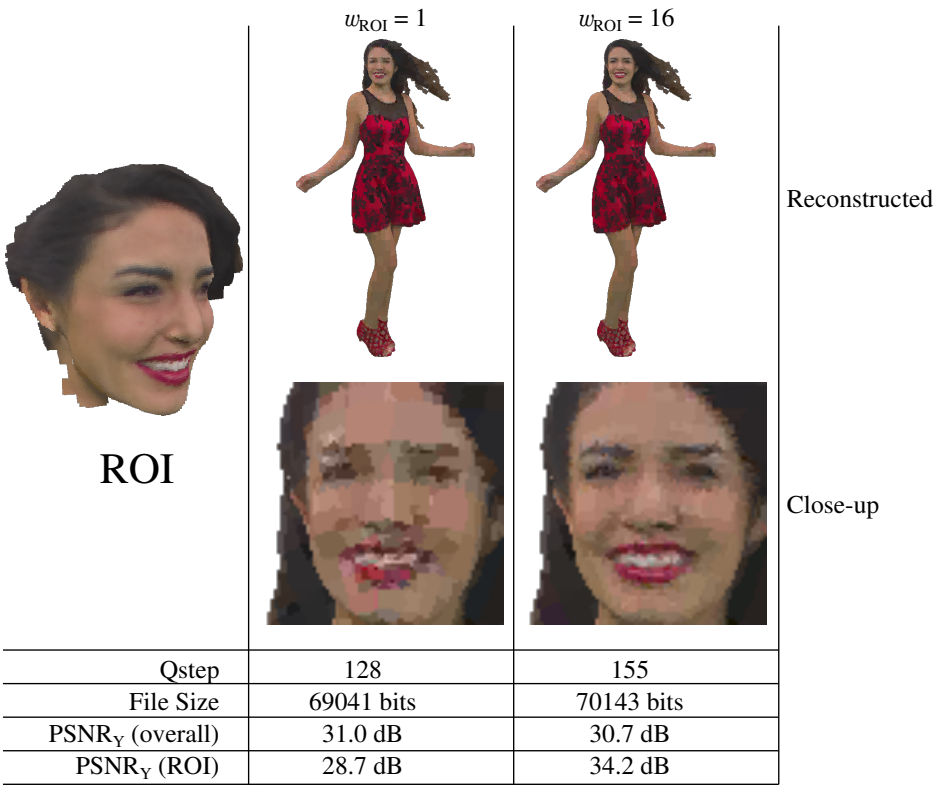


Figure 6.6: Average rate-distortion curves for the sequence Loot using the weighted PSNR



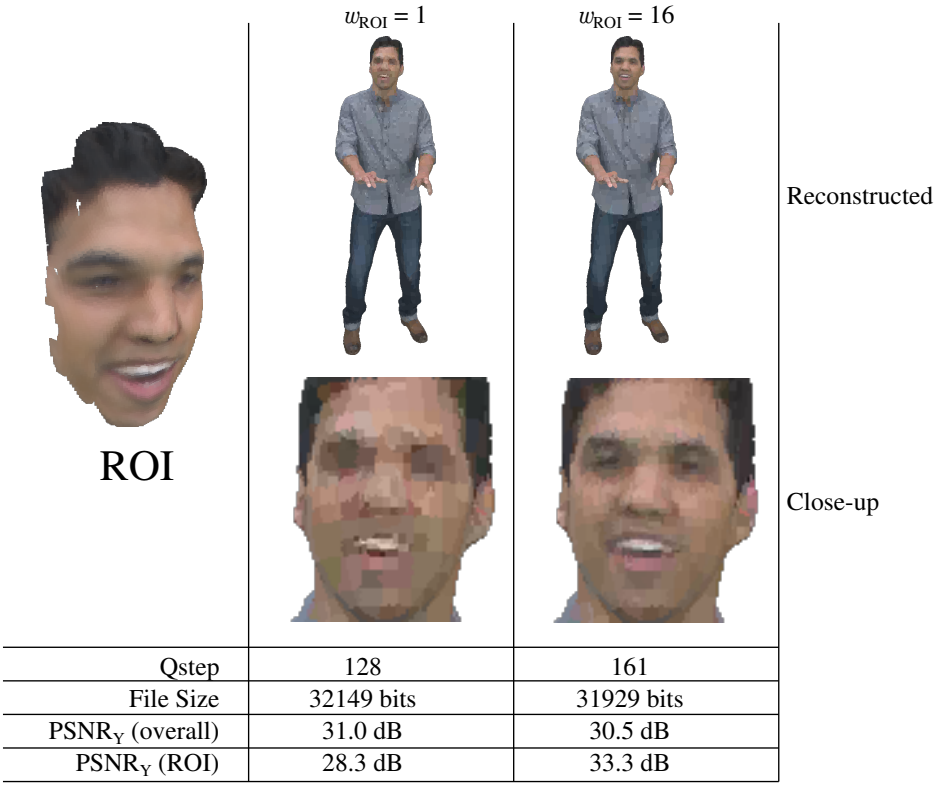
Qstep	128	133
File Size	150006 bits	148974 bits
$PSNR_Y$ (overall)	25.8 dB	25.7 dB
$PSNR_Y$ (ROI)	28.4 dB	34.1 dB

Figure 6.7: A frame of the point cloud Longdress encoded with w_{ROI} equals to 1 and 16. The quantization step size was adjusted to yield similar encoded file sizes. A close-up of the reconstructed point cloud is also shown.



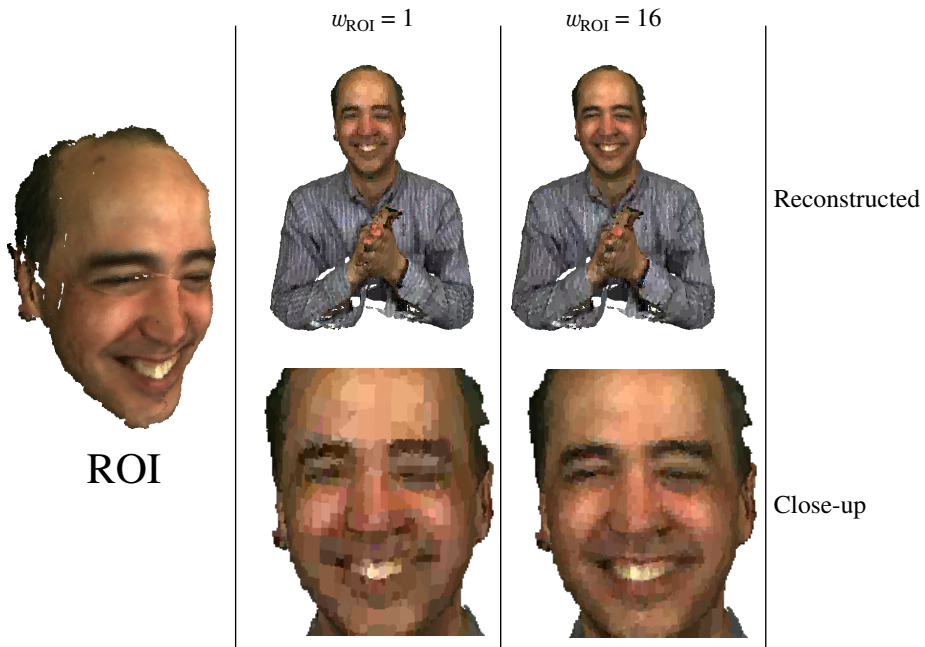
Qstep	128	155
File Size	69041 bits	70143 bits
$PSNR_Y$ (overall)	31.0 dB	30.7 dB
$PSNR_Y$ (ROI)	28.7 dB	34.2 dB

Figure 6.8: A frame of the point cloud Redandblack encoded with w_{ROI} equals to 1 and 16. The quantization step size was adjusted to yield similar encoded file sizes. A close-up of the reconstructed point cloud is also shown.



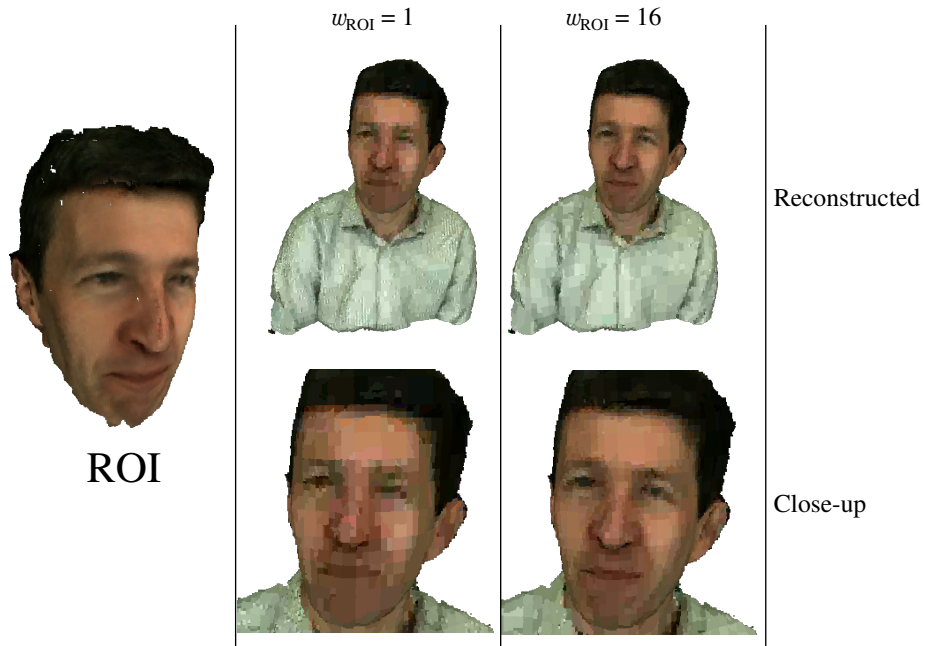
Qstep	128	161
File Size	32149 bits	31929 bits
$PSNR_Y$ (overall)	31.0 dB	30.5 dB
$PSNR_Y$ (ROI)	28.3 dB	33.3 dB

Figure 6.9: A frame of the point cloud Man encoded with w_{ROI} equals to 1 and 16. The quantization step size was adjusted to yield similar encoded file sizes. A close-up of the reconstructed point cloud is also shown.



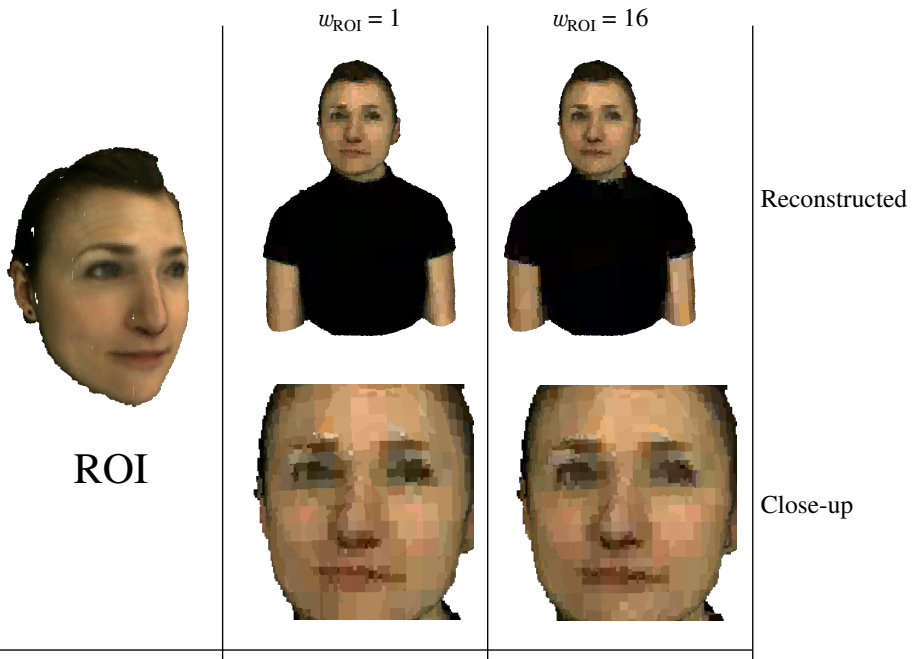
Qstep	128	197
File Size	47316 bits	47647 bits
PSNR_Y (overall)	26.3 dB	25.0 dB
PSNR_Y (ROI)	28.2 dB	31.9 dB

Figure 6.10: A frame of the point cloud Phil encoded with w_{ROI} equals to 1 and 16. The quantization step size was adjusted to yield similar encoded file sizes. A close-up of the reconstructed point cloud is also shown.



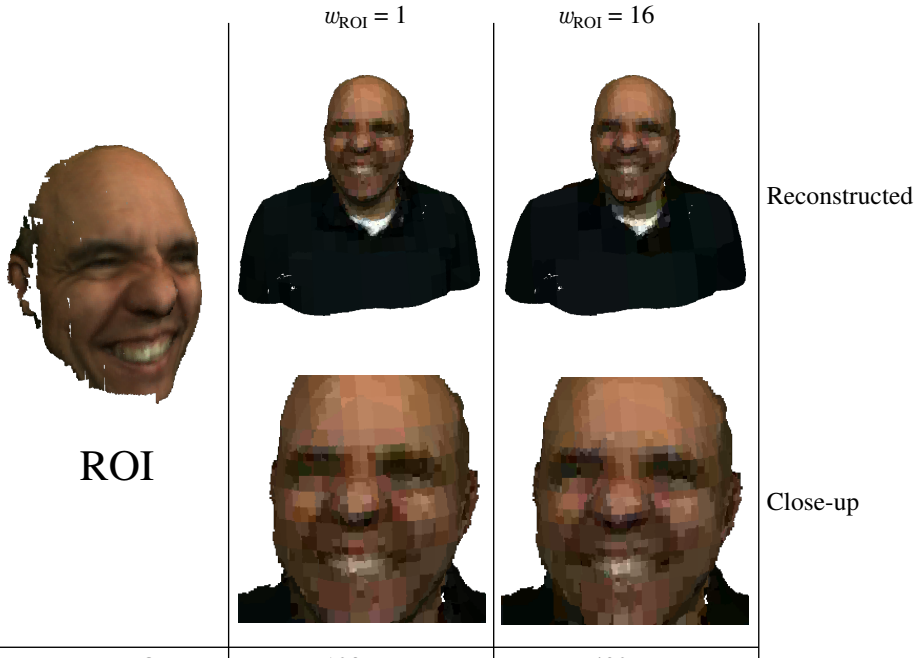
Qstep	128	206
File Size	33192 bits	33374 bits
PSNR_Y (overall)	25.2 dB	24.4 dB
PSNR_Y (ROI)	29.4 dB	32.9 dB

Figure 6.11: A frame of the point cloud Andrew encoded with w_{ROI} equals to 1 and 16. The quantization step size was adjusted to yield similar encoded file sizes. A close-up of the reconstructed point cloud is also shown.



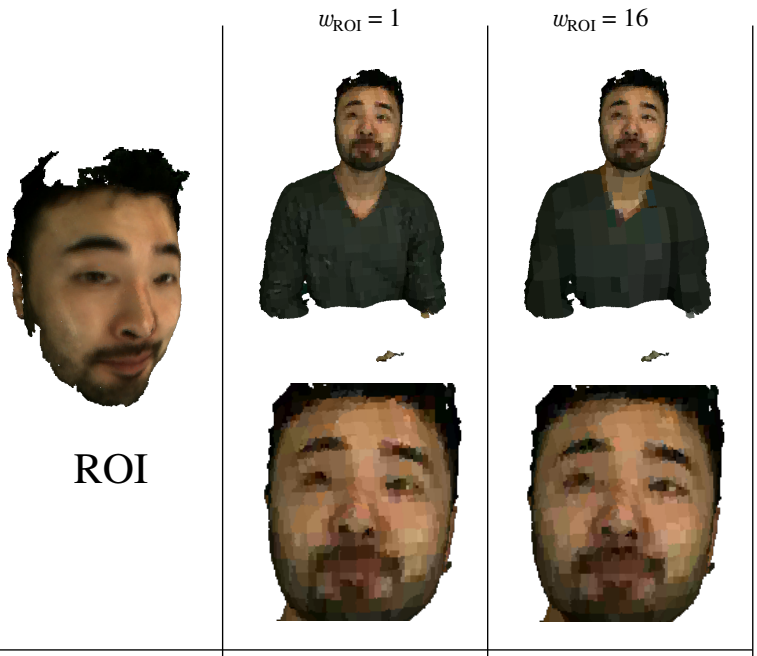
Qstep	128	401
File Size	14546 bits	14637 bits
PSNR_{Y} (overall)	32.5 dB	29.9 dB
PSNR_{Y} (ROI)	29.0 dB	30.0 dB

Figure 6.12: A frame of the point cloud Sarah encoded with w_{ROI} equals to 1 and 16. The quantization step size was adjusted to yield similar encoded file sizes. A close-up of the reconstructed point cloud is also shown.



Qstep	128	402
File Size	10675 bits	10690 bits
PSNR_{Y} (overall)	32.5 dB	30.0 dB
PSNR_{Y} (ROI)	29.0 dB	30.8 dB

Figure 6.13: A frame of the point cloud Ricardo encoded with w_{ROI} equals to 1 and 16. The quantization step size was adjusted to yield similar encoded file sizes. A close-up of the reconstructed point cloud is also shown.



Qstep	128	430
File Size	14833 bits	14802 bits
PSNR_{Y} (overall)	31.8 dB	29.8 dB
PSNR_{Y} (ROI)	29.1 dB	29.8 dB

Figure 6.14: A frame of the point cloud David encoded with w_{ROI} equals to 1 and 16. The quantization step size was adjusted to yield similar encoded file sizes. A close-up of the reconstructed point cloud is also shown.

6.5 CONCLUSION

We introduced a framework for ROI identification using multiple projections and image identification algorithms. The multiple projections are obtained by varying the azimuth and elevation angle. We chose faces as ROI because of their importance in subjective quality. However, any other object can be defined as ROI. By modifying the distortion measure to a weighted-distortion measure, we introduce ROI coding with RAHT transform coding. However, any other point cloud coding method that optimizes the weighted distortion measure could be used. The results show an improvement in the ROI PSNR, controlled by the weight attributed to ROI. Higher weights result in better quality. The gain in quality was found to be, on average, 1.03, 2.47, 3.97, 5.53, and 7.16 dB when the weight attributed to the ROI was 2, 4, 8, 16 and 32, respectively. The relatively small ROI sizes allow for a significant improvement of ROI voxels at the expense of a minimal degradation to non-ROI voxels. The real motivation is the improved subjective quality of the ROI-coded point clouds, as viewers often prefer to preserve the face quality and have a hard time noticing a slightly higher distortion in textured areas of little interest.

7 CONCLUSIONS

Based on the original proposal of the Region-Adaptive Hierarchical Transform, a transform that has almost linear complexity and can be easily computed in real-time, we have proposed four modifications to improve its performance, guarantee reproducibility, and to add the capability to compress regions-of-interest and plenoptic point clouds.

In Chapter 3 we worked on the the entropy coder. RAHT generates a sequence of coefficients that are rastered in a vector, quantized, and entropy coded. We have observed that RAHT performance can be improved by the choice of how coefficients are ordered in this vector while using RLGR, an adaptive encoder. RLGR switches between two coding modes: Golomb-Rice and Run-Length. This encoder uses two parameters that are backward adapted as coefficients are encoded. One of these parameters adapts to the coefficients amplitudes and the other is a criteria to switch between Run-Length and Golomb-Rice.

RAHT transform is represented by a binary tree and coefficients can be ordered according to a tree searching algorithm or by trying to estimate coefficients amplitudes. We have presented and tested four sorting criteria: breadth-first scanning, depth-first transversal scanning, weight-sorted, and variance-sorted. The first two are tree scanning algorithms. Weight-sorted is based upon the fact that every RAHT coefficient has a weight associated to it that have been observed to be correlated to coefficient amplitude. Variance is also correlated to amplitude and can be estimated for every coefficient.

Breadth-first scanning presented the best performance among the four tested sorting criteria. The performance was competitive, surpassing the one from the original work, and even outperforming the much more complex GT-based point cloud coder.

In order to guarantee reproducibility, we have introduced a new RAHT definition in Chapter 4 that allows for fixed-point (integer) implementation without impacting the compression performance. RAHT is originally based on natural operations that are performed with floating-point arithmetic. We have replaced all operations by integer equivalents and, from matrix decomposition, have simplified the forward and inverse transform. This can effectively change the RAHT coder description in the context of point cloud compression standardization, and it may enable more players in the area.

Fixed-point arithmetic accuracy depends on how many bits are used to represent the fractional part of a number, referred as precision bits. Results show that as low as 8-bit fixed-point precision may be good enough for our compression applications, and any representation using 10 bits and above yields negligible compression performance impact.

In Chapter 5 we have extended point clouds representation to encompass multiple-color voxels, which are samples of the plenoptic function for each voxel, thus referring to them as plenoptic point clouds. We believe that plenoptic point clouds can better represent the real-world because

objects with higher specularity may be less accurately represented by traditional single-color point clouds. We have developed and tested four methods to extend the RAHT coder to compress plenoptic point clouds. Two of the proposed approaches extend the RAHT over the sphere representing the plenoptic function (RAHT-1 and RAHT-2), while two other approaches involve the combination of RAHT with 1D transforms over the cameras color vector (RAHT-KLT and RAHT-DCT). Among all four proposed methods, RAHT-KLT presents the best overall performance, even when the objects in the scene varies from completely specular to plainly diffuse.

Finally, in Chapter 6 we modified RAHT to allow for region-of-interest compression where ROI quality is preserved in detriment to points that are not in the ROI. ROI coding was introduced by modifying the distortion measure to a weighted-distortion measure, *i. e.*, the distortion measure is made more susceptible to artifacts in the ROI. The results show an improvement in the ROI PSNR, controlled by the weight attributed to ROI. Higher weights result in better quality. The gain in quality was found to be, on average, 1.03, 2.47, 3.97, 5.53, and 7.16 dB when the weight attributed to the ROI was 2, 4, 8, 16 and 32, respectively. The relatively small ROI sizes allow for a significant improvement of ROI voxels at the expense of a minimal degradation to non-ROI voxels.

The real motivation is the improved subjective quality of the ROI-coded point clouds, as viewers often prefer to preserve the face quality and have a hard time noticing a slightly higher distortion in textured areas of little interest.

7.1 SUMMARY OF CONTRIBUTIONS

Our contribution to the entropy (Chapter 3) coder arose with the breadth-first scanning. It was noted, almost by chance, that using this scanning resulted in fewer bits for the encoding of quantized coefficients when combined with RLGR. We later discovered the work of Chou *et al.* [62], who also noticed that the performance could be improved by rearranging the coefficients. They proposed ordering the coefficients based on their weights. We have also proposed to order coefficients based on their estimated variance, as we believed it could be an estimator of coefficient amplitude.

The implementation of the fixed-point RAHT (Chapter 4) was proposed by us, along with the modification in the RAHT transform to accommodate it. We have also proposed the algorithm to efficiently compute the squared root of a number using integer arithmetic and how to compute an initial guess efficiently.

In Chapter 5 we have crafted and tested four methods to extend the RAHT coder to compress plenoptic point clouds and in Chapter 6 we introduced the framework for ROI identification using multiple projections the modification of the distortion measure to a weighted-distortion measure.

7.2 FUTURE WORK

As future work, much is still to be done for improving the coder, specially with regards to entropy coding. For the plenoptic point clouds we need to identify regions of high and low specularity that could be treated differently. We can also encode plenoptic point clouds taking advantage of video codecs, generating one texture map for each camera.

Optimization of ROI weights using perceptual studies can also be applied. This would extend ROI coding to multi-level weights (e.g., from saliency maps), optimizing the side information, and applying ROI coding to point cloud geometry.

7.3 PUBLICATIONS

I **Compression of Plenoptic Point Cloud** [76]

Published on IEEE Transactions on Image Processing journal. Covers the topic presented in Chapter 5.

II **Identification and Encoding of Regions of Interest in Point Clouds**

Paper submitted to the IEEE Transactions on Image Processing journal. Describes the ROI compression presented in Chapter 6. (authors: G. Sandri, V. F. Figueiredo, R. L. de Queiroz, P. A. Chou).

III **Integer Alternative for the Region-Adaptive Hierarchical Transform** [66]

Paper published in the IEEE Signal Processing Letters. Describes the integer implementation of RAHT of chapter 4.

IV **Compression of Plenoptic Point Clouds Using the Region-Adaptive Hierarchical Transform** [77]

Paper presented in the International Conference on Image Processing, 2018, held in Athens, Greece. Presents the methods RAHT-1, face crossing point and sphere crossing point presented in Chapter 5.

V **Point Cloud compression incorporating region of interest coding** [105]

Paper presented in the International Conference on Image Processing, 2019, held in Taipei, Taiwan. Covers the topic of Chapter 6.

VI **Representação Compressível para Codificação de Point Clouds Plenópticas** [78]

Presented at the XXXVI Brazilian Symposium on Telecommunications and Signal Processing held in Campina Grande, Brazil. Presents the improvements on RAHT described in chapter 3 and how to compress plenoptic point clouds using the cylindrical projection described in Chapter 5.

VII **Compressão de Nuvem de Pontos Incorporando Codificação de Região de Interesse** [106]

Paper presented at the XXXVII Brazilian Symposium on Telecommunications and Signal Processing, held in Petrópolis, Brazil. Covers the topic of Chapter 6. This paper received the best scientific paper award in this conference.

VIII **Comments on "Compression of 3D Point Clouds Using a Region-Adaptive Hierarchical Transform"** [36]

Published on arXiv on May 2018. Describes the sorting criterias proposed in chapter 3.

IX **Fixed-point version of RAHT with rate-distortion impact tests** [67]

Input document M44486 submitted to the International Organization for Standardization (ISO/IEC JTC1/SC29/WG11). Covers the integer implementation of RAHT described in chapter 4.

X **Signaling view dependent point cloud information by SEI** [79]

Input document M43598 submitted to the International Organization for Standardization (ISO/IEC JTC1/SC29/WG11). Based on the paper of Sandri *et. al.* [76], proposes the implementation of SEI (Additional Supplemental Enhancement Information) to encode the plenoptic information in a transparent way. A decoder not able to decode a plenoptic point cloud would still be able to decode a single color point cloud where the color associated to each voxel is the average of the colors captured by the cameras.

XI **Transform Coder for View-dependent Point Cloud Attributes** [80]

Input document M43596 submitted to the International Organization for Standardization (ISO/IEC JTC1/SC29/WG11). Proposes the implementation of the plenoptic point cloud compression as described by Sandri *et. al.* [76].

REFERENCES

[1] A. Albar and M. R. Swash, "Portable holoscopic 3d camera adaptor for raspberry pi," in *Digital Media Industry Academic Forum*, July 2016, pp. 185–188.

[2] B. Pal, S. Khaiyum, and Y. S. Kumaraswamy, "3d point cloud generation from 2d depth camera images using successive triangulation," in *International Conference on Innovative Mechanisms for Industry Applications*, February 2017, pp. 129–133.

[3] S. Schwarz *et al.*, "Emerging MPEG standards for point cloud compression," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 1, pp. 133–148, March 2019.

[4] MPEG, "Mpeg121 version of MPEG standardisation roadmap," ISO/IEC JTC1/SC29/WG11 MPEG, output document N17332, January 2017.

[5] 3D Graphics Subgroup, "Call for proposals for point cloud compression v2," ISO/IEC JTC1/SC29/WG11 MPEG, output document N16763, April 2017.

[6] R. Skarbez, F. P. Brooks Jr., and M. C. Whitton, "A survey of presence and related concepts," *ACM Computing Surveys*, vol. 50, no. 6, pp. 96:1–96:39, January 2018.

[7] L. Pei and Z. Rui, "The analysis of stereo vision 3d point cloud data of autonomous vehicle obstacle recognition," in *International Conference on Intelligent Human-Machine Systems and Cybernetics*, August 2015, vol. 2, pp. 207–210.

[8] D. Priyasad *et al.*, "Point cloud based autonomous area exploration algorithm," in *Moratuwa Engineering Research Conference*, May 2018, pp. 318–323.

[9] J. Ping, Y. Liu, and D. Weng, "Comparison in depth perception between virtual reality and augmented reality systems," in *IEEE Conference on Virtual Reality and 3D User Interfaces*, March 2019, pp. 1124–1125.

[10] J. Schild *et al.*, "Applying multi-user virtual reality to collaborative medical training," in *IEEE Conference on Virtual Reality and 3D User Interfaces*, March 2018, pp. 775–776.

[11] P. Rajeswaran, T. Kesavadas, P. Jani, and P. Kumar, "AirwayVR: Virtual reality trainer for endotracheal intubation-design considerations and challenges," in *IEEE Conference on Virtual Reality and 3D User Interfaces*, March 2019, pp. 1130–1131.

[12] T. Kanatani, H. Kume, T. Taketomi, T. Sato, and N. Yokoya, "Detection of 3d points on moving objects from point cloud data for 3d modeling of outdoor environments," in *IEEE International Conference on Image Processing*, September 2013, pp. 2163–2167.

[13] R. Szliski, *Computer Vision: Algorithms and Applications*, Springer, 2011.

[14] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, Prentice Hall, second edition, 2002.

[15] D. A. Forsyth and J. ponce, *Computer Vision: A Modern Approach*, Prentice Hall, second edition, 2011.

[16] V. Vezhnevets, V. Sazonov, and A. Andreeva, "A survey on pixel-based skin color detection techniques," *Procedings of GRAPHICON*, pp. 85–92, 2003.

[17] International Telecommunication Union, *Recommendation ITU-R BT.601-7: Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios*, broadcasting service series edition, March 2011.

[18] M. Berger *et al.*, "A survey of surface reconstruction from point clouds," *Computer Graphics Forum*, vol. 36, pp. 301–329, January 2017.

[19] J. Peng, C.-S. Kim, and C.-C. Jay Kuo, "Technologies for 3d mesh compression: A survey," *Journal of Visual Communication and Image Representation*, vol. 16, no. 6, pp. 688–733, 2005.

[20] L. Goode, "Are holograms the future of how we capture memories," The Verge, November 2017.

[21] N. Namitha, S.M. Vaitheeswaran, V.K. Jayasree, and M.K. Bharat, "Point cloud mapping measurements using Kinect RGB-D sensor and Kinect Fusion for visual odometry," *Procedia Computer Science*, vol. 89, pp. 209—212, 2016.

[22] W. Tao, Y. Lei, and P. Mooney, "Dense point cloud extraction from UAV captured images in forest area," in *IEEE International Conference on Spatial Data Mining and Geographical Knowledge Services*, June 2011, pp. 389–392.

[23] M. Berk, O. Schubert, H. Kroll, B. Buschardt, and D. Straub, "Exploiting redundancy for reliability analysis of sensor perception in automated driving vehicles," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–13, 2019.

[24] T. Ochotta and D. Saupe, "Compression of point-based 3d models by shape-adaptive wavelet coding of multi-height fields," in *Proceedings of the First Eurographics Conference on Point-Based Graphics*. 2004, pp. 103–112, Eurographics Association.

[25] R. Schnabel and R. Klein, "Octree-based point-cloud compression," in *Proceedings of the 3rd Eurographics / IEEE VGTC Conference on Point-Based Graphics*. 2006, pp. 111–121, Eurographics Association.

[26] Y. Huang, J. Peng, C. C. J. Kuo, and M. Gopi, "A generic scheme for progressive point cloud coding," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 2, pp. 440–453, March 2008.

[27] C. Loop, C. Zhang, and A. Zhang, "Real-time high-resolution sparse voxelization with application to image-based modeling," in *Proceedings of the 5th High-Performance Graphics Conference*, New York, NY, USA, 2013, HPG '13, pp. 73–79, ACM.

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

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

[30] D. C. Garcia and R. L. de Queiroz, "Context-based octree coding for point-cloud video," in *IEEE International Conference on Image Processing*, September 2017, pp. 1412–1416.

[31] D. C. Garcia and R. L. de Queiroz, "Intra-frame context-based octree coding for point-cloud geometry," in *IEEE International Conference on Image Processing*, October 2018, pp. 1807–1811.

[32] D. Tian, H. Ochimizu, C. Feng, R. Cohen, and A. Vetro, "Geometric distortion metrics for point cloud compression," in *IEEE International Conference on Image Processing*, September 2017, pp. 3460–3464.

[33] MPEG, "Common test conditions for point cloud compression," ISO/IEC JTC1/SC29/WG11 MPEG, output document N17766, July 2018.

[34] E. Alexiou *et al.*, "A comprehensive study of the rate-distortion performance in MPEG point cloud compression," *APSIPA Transactions on Signal and Information Processing*, vol. 8, pp. e27, 2019.

[35] C. Zhang, D. Florêncio, and C. Loop, "Point cloud attribute compression with graph transform," in *IEEE International Conference on Image Processing*, October 2014, pp. 2066–2070.

[36] G. Sandri, R. L. de Queiroz, and P. A. Chou, "Comments on "compression of 3d point clouds using a region-adaptive hierarchical transform"," *arXiv:1805.09146*, May 2018.

[37] D. B. West, *Introduction to Graph Theory*, Prentice Hall, second edition, September 2000.

[38] N. Kuroki, T. Manabe, and M. Numa, "Adaptive arithmetic coding for image prediction errors," in *IEEE International Symposium on Circuits and Systems*, May 2004, vol. 3, pp. III–961.

[39] P. G. Howard and J. S. Vitter, *Practical Implementations of Arithmetic Coding*, pp. 85–112, Springer US, 1992.

[40] P. A. Chou and R. L. de Queiroz, "Gaussian process transforms," in *IEEE International Conference on Image Processing*, September 2016, pp. 1524–1528.

[41] R. L. de Queiroz and P. A. Chou, "Transform coding for point clouds using a gaussian process model," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3507–3517, July 2017.

[42] G. Strang and T. Nguyen, *Wavelets and filter banks*, Wellesley-Cambridge Press, 1997.

[43] MPEG, "PCC test model category 2 v0," ISO/IEC JTC1/SC29/WG11 MPEG, output document N17248, October 2017.

[44] S.-Y. Chien *et al.*, "Hardware architecture design of video compression for multimedia communication systems," *IEEE Communications Magazine*, vol. 43, pp. 122–131, September 2005.

[45] H. Hoppe *et al.*, "Surface reconstruction from unorganized points," *ACM SIGGRAPH Computer Graphics*, vol. 26, no. 2, pp. 71–78, July 1992.

[46] M. Harris, "Focusing on everything," *IEEE Spectrum*, vol. 49, no. 5, pp. 44–50, May 2012.

[47] E. Adelson and J. Bergen, "The plenoptic function and the elements of early vision," *Computational Models of Visual Processing*, pp. 3–20, 1991.

[48] G. Lippmann, "Épreuves réversibles donnant la sensation du relief," *Journal de Pshysique Théorique et Appliquée*, vol. 7, no. 1, pp. 821–825, 1908.

[49] T. Georgeiv *et al.*, "Spatio-angular resolution tradeoffs in integral photography," in *Eurographics Conference on Rendering Techniques*. 2006, pp. 263–272, Eurographics Association.

[50] T. Georgiev, S. Tambe, A. Lumsdaine, J. Gille, and A. Veeraraghavan, "The radon image as plenoptic function," in *IEEE International Conference on Image Processing*, October 2014, pp. 1922–1926.

[51] N. Sabater, M. Seifi, V. Drazic, G. Sandri, and P. Pérez, *Accurate Disparity Estimation for Plenoptic Images*, pp. 548–560, Springer International Publishing, 2015.

[52] A. Saxena, J. Schulte, and A. Y. Ng, "Depth estimation using monocular and stereo cues," in *International Joint Conference on Artifical Intelligence*. 2007, IJCAI'07, pp. 2197–2203, Morgan Kaufmann Publishers Inc.

[53] N. Grammalidis and M. G. Strintzis, "Disparity and occlusion estimation in multiocular systems and their coding for the communication of multiview image sequences," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 3, pp. 328–344, June 1998.

[54] R. Ng, *Digital Light Field Photography*, Ph.D. thesis, Stanford, Stanford, CA, USA, 2006.

[55] A. Isaksen, L. McMillan, and S. J. Gortler, "Dynamically reparameterized light fields," in *Annual Conference on Computer Graphics and Interactive Techniques*, 2000, pp. 297–306.

[56] F. P. Nava and J. P. Luke, "Simultaneous estimation of super-resolved depth and all-in-focus images from a plenoptic camera," in *3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video*, May 2009, pp. 1–4.

[57] T. E. Bishop and P. Favaro, "Plenoptic depth estimation from multiple aliased views," in *IEEE International Conference on Computer Vision Workshops*, September 2009, pp. 1622–1629.

[58] T. E. Bishop and P. Favaro, "The light field camera: Extended depth of field, aliasing, and superresolution," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 5, pp. 972–986, May 2012.

[59] A. G. Asuero, A. Sayago, and G. González, "The correlation coefficient: An overview," *Critical Reviews in Analytical Chemistry*, vol. 36, pp. 41–59, January 2006.

[60] G. Bjøntegaard, "Calculation of average PSNR differences between RD-curves," Visual Coding Experts Group, ITU-T Q6/16 document VCEG-M33, April 2001.

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

[62] P. A. Chou, E. Pavez, R. L. de Queiroz, and A. Ortega, "Dynamic polygon clouds: Representation and compression for VR/AR," Tech. Rep. MSR-TR-2016-59, Microsoft Research, January 2017.

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

[64] J. Navarrete *et al.*, "3DCOMET: 3d compression methods test dataset," *Robotics and Autonomous Systems*, vol. 75, pp. 550–557, 2016.

[65] IEEE 754-2019, "IEEE Standard for Floating-Point Arithmetic," June 2019.

[66] G. Sandri, P. A. Chou, M. Krivokuća, and R. L. de Queiroz, "Integer alternative for the region-adaptive hierarchical transform," *IEEE Signal Processing Letters*, vol. 26, no. 9, pp. 1369–1372, September 2019.

[67] G. Sandri *et al.*, "Fixed-point version of RAHT with rate-distortion impact tests," ISO/IEC JTC1/SC29/WG11 MPEG, input document M44486, October 2018.

[68] H. Anton and C. Horres, *Elementary Linear Algebra: applications version*, Wiley, 11 edition, 2013.

[69] D. Bindel, J. Demmel, W. Kahan, and O. Marques, "On computing Givens rotations reliably and efficiently," Tech. Rep. CS-00-448, University of Tennessee, October 2000.

[70] E. d'Eon, B. Harrison, T. Myers, and P. A. Chou, "8i voxelized full bodies - a voxelized point cloud dataset," ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document WG11M40059/WG1M74006, January 2017.

[71] S. Orts-Escolano *et al.*, "Holoportation: Virtual 3d teleportation in real-time," in *Annual Symposium on User Interface Software and Technology*. 2016, pp. 741–754, ACM.

[72] A. Pujol-Miro, J. Ruiz-Hidalgo, and J. R. Casas, "Registration of images to unorganized 3d point clouds using contour cues," in *European Signal Processing Conference*, August 2017, pp. 81–85.

[73] C. Perra, F. Murgia, and D. Giusto, "An analysis of 3d point cloud reconstruction from light field images," in *International Conference on Image Processing Theory, Tools and Applications*, December 2016, pp. 1–6.

[74] D. Thanou, P. A. Chou, and P. Frossard, "Graph-based motion estimation and compensation for dynamic 3d point cloud compression," in *IEEE International Conference on Image Processing*, september 2015, pp. 3235–3239.

[75] J. Kammerl *et al.*, "Real-time compression of point cloud streams," in *IEEE International Conference on Robotics and Automation*, May 2012, pp. 778–785.

[76] G. Sandri, R. L. de Queiroz, and P. A. Chou, "Compression of plenoptic point clouds," *IEEE Transactions on Image Processing*, vol. 28, no. 3, pp. 1419–1427, March 2019.

[77] G. Sandri, R. L. de Queiroz, and P. A. Chou, "Compression of plenoptic point clouds using the region-adaptive hierarchical transform," in *IEEE International Conference on Image Processing*, October 2018, pp. 1153–1157.

[78] G. Sandri, R. L. de Queiroz, and P. A. Chou, "Representação compressível para codificação de point clouds plenópticas," in *XXXVI Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*, September 2018.

[79] G. Sandri, R. L. de Queiroz, and P. A. Chou, "Signaling view dependent point cloud information by SEI," ISO/IEC JTC1/SC29/WG11 MPEG, input document M43598, July 2018.

[80] R. L. de Queiroz, G. Sandri, and P. A. Chou, "Transform coder for view-dependent point cloud attributes," ISO/IEC JTC1/SC29/WG11 MPEG, input document M43596, July 2018.

[81] D. Liu, L. Wang, L. Li, Zhiwei Xiong, Feng Wu, and Wenjun Zeng, "Pseudo-sequence-based light field image compression," in *IEEE International Conference on Multimedia Expo Workshops*, July 2016, pp. 1–4.

[82] S. Zhao, Z. Chen, K. Yang, and H. Huang, "Light field image coding with hybrid scan order," in *Visual Communications and Image Processing*, November 2016, pp. 1–4.

[83] L. Li, Z. Li, B. Li, D. Liu, and H. Li, "Pseudo sequence based 2-d hierarchical coding structure for light-field image compression," in *Data Compression Conference*, April 2017, pp. 131–140.

[84] C. Jia *et al.*, "Light field image compression with sub-apertures reordering and adaptive reconstruction," in *Advances in Multimedia Information Processing*, 2018, pp. 47–55.

[85] C. Jia *et al.*, "Optimized inter-view prediction based light field image compression with adaptive reconstruction," in *IEEE International Conference on Image Processing*, September 2017, pp. 4572–4576.

[86] Y. Li, M. Sjöström, R. Olsson, and U. Jennehag, "Efficient intra prediction scheme for light field image compression," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2014, pp. 539–543.

[87] Y. Li, R. Olsson, and M. Sjöström, "Compression of unfocused plenoptic images using a displacement intra prediction," in *IEEE International Conference on Multimedia Expo Workshops*, July 2016, pp. 1–4.

[88] Y. Li, M. Sjöström, R. Olsson, and U. Jennehag, "Coding of focused plenoptic contents by displacement intra prediction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 7, pp. 1308–1319, July 2016.

[89] Y. Li, M. Sjöström, R. Olsson, and U. Jennehag, "Scalable coding of plenoptic images by using a sparse set and disparities," *IEEE Transactions on Image Processing*, vol. 25, no. 1, pp. 80–91, January 2016.

[90] C. Conti, P. Nunes, and L. D. Soares, "Hevc-based light field image coding with bi-predicted self-similarity compensation," in *IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, July 2016, pp. 1–4.

[91] R. Monteiro *et al.*, "Light field HEVC-based image coding using locally linear embedding and self-similarity compensated prediction," in *IEEE International Conference on Multimedia Expo Workshops*, July 2016, pp. 1–4.

[92] X. Zhang *et al.*, "Surface light field compression using a point cloud codec," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 1, pp. 163–176, March 2019.

[93] D. N. Wood *et al.*, "Surface light fields for 3d photography," in *Annual Conference on Computer Graphics and Interactive Techniques*, New York, NY, USA, 2000, pp. 287–296, ACM Press/Addison-Wesley Publishing Co.

[94] W. Chen, J. Bouguet, M. H. Chu, and R. Grzeszczuk, "Light field mapping: Efficient representation and hardware rendering of surface light fields," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 447–456, July 2002.

[95] G. Turk and M. Levoy, "Zippered polygon meshes from range images," *Proceedings of Computer Graphics*, 1996.

[96] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," *Proceedings of Computer Graphics*, 1996.

[97] A. Torrence, "Martin newell's original teapot," *Proceedings of Computer Graphics*, 2006.

[98] B. T. Phong, "Illumination for computer generated pictures," *Communications of the ACM*, vol. 18, no. 6, pp. 311–317, June 1975.

[99] T. Leisti, J. Radun, T. Virtanen, R. Halonen, and G. Nyman, "Subjective experience of image quality: attributes, definitions, and decision making of subjective image quality," in *Image Quality and System Performance*. 2009, vol. 7242, pp. 130–138, SPIE.

[100] H. Hadizadeh and I. V. Bajić, "Saliency-aware video compression," *IEEE Transactions on Image Processing*, vol. 23, no. 1, pp. 19–33, January 2014.

[101] M. Krivokuća, M. Koroteev, and P. A. Chou, "A volumetric approach to point cloud compression," arXiv:1810.00484 [eess.IV], September 2018.

[102] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2325–2383, October 1998.

[103] T. Linder and R. Zamir, "High-resolution source coding for non-difference distortion measures: the rate-distortion function," *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 533–547, March 1999.

[104] J. Li, N. Chaddha, and R. M. Gray, "Asymptotic performance of vector quantizers with a perceptual distortion measure," *IEEE Transactions on Information Theory*, vol. 45, no. 4, pp. 1082–1091, May 1999.

[105] G. Sandri, V. F. Figueiredo, P. A. Chou, and R. L. de Queiroz, "Point cloud compression incorporating region of interest coding," in *IEEE International Conference on Image Processing*, September 2019, pp. 4370–4374.

[106] G. Sandri, V. F. Figueiredo, P. A. Chou, and R. L. de Queiroz, "Compressão de nuvem de pontos incorporando codificação de região de interesse," in *XXXVII Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*, Petrópolis, Brazil, September 2019.

[107] Z. Wen, Y. Yan, and H. Cui, "Study on segmentation of 3d human body based on point cloud data," in *International Conference on Intelligent System Design and Engineering Application*, January 2012, pp. 657–660.

[108] M. Qiao, J. Cheng, W. Bian, and D. Tao, "Biview learning for human posture segmentation from 3d points cloud," *PLoS One*, vol. 9, no. 1, January 2014.

[109] P. Mandikal, K. L. Navaneet, and R. V. Babu, "3D-PSRNet: Part segmented 3d point cloud reconstruction from a single image," *Computing Research Repository*, vol. abs/1810.00461, 2018.

[110] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, December 2001, vol. 1, pp. I–I.

[111] G. M. Morton, "A computer oriented geodetic data base and a new technique in file sequencing," Tech. Rep., IBM Ltd., Ottawa 4, Ontario, Canada, March 1966.

[112] H. P. Hariharan, T. Lange, and T. Herfet, "Low complexity light field compression based on pseudo-temporal circular sequencing," in *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*, June 2017, pp. 1–5.

[113] A. J. Lipton, H. Fujiyoshi, and R. S. Patil, "Moving target classification and tracking from real-time video," in *IEEE Workshop on Applications of Computer Vision*, October 1998, pp. 8–14.

[114] R. Khoshabeh, S. H. Chan, and T. Q. Nguyen, "Spatio-temporal consistency in video disparity estimation," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2011, pp. 885–888.

APPENDIX

I. DATABASE

The first dataset is the Microsoft voxelized upper bodies [63] (MVUB). This dataset contains dynamic voxelized point cloud sequences of the the upper bodies of 5 subjects, depicted in Figure I.1. The upper bodies of these subjects are captured by four frontal RGBD cameras, at 30 fps, over a 7–10 s period for each. It was submitted as an input for the MPEG/JPEG to be used for validation and testing. Two spatial resolutions are provided for each sequence: a cube of $512 \times 512 \times 512$ voxels (depth of 9) and a cube of $1024 \times 1024 \times 1024$ voxels (depth of 10). In each cube, only voxels near the surface of the subjects are occupied. The attributes of a voxel are the red, green, and blue components of the surface color. Voxels at depth 9 are approximately 1.5 mm cubed, while voxels at depth 10 are approximately 0.75 mm cubed.

(a) Andrew (b) Ricardo (c) Phil (d) David (e) Sarah

Figure I.1: Rendered views of the Microsoft voxelized upper bodies dataset.

The second dataset is more recent, submetted by 8i to help in the MPEG/JPEG efforts and comprises voxelized point clouds of full bodies [70] (8iVFB). In each sequence, the full body of a human subject is captured by 42 RGB cameras configured in 14 clusters (each cluster acting as a logical RGBD camera), at 30 fps, over a 10 s period. The point clouds were directly sent to

us and were not made available online. However, they provide a voxelized versions of the point clouds at the JPEG Pleno Database [1] with a depth of 10. In ours experiments, we voxelized the point clouds with a depth of 11. Beyond the average color captured by each camera, we also have available the color as captured by each pod individually. This information is used for ours tests with plenoptic point clouds. A rendered view of these images is given in Figure I.2.

(a) Boxer (b) Longdress (c) Loot (d) Soldier (e) Redandblack (f) Thaidancer

Figure I.2: Rendering of a view of the 8i voxelized full bodies dataset

Table I.1 summarizes the number of points, voxels after voxelization and cameras employed for capture for each cloud in the 8i voxelized full bodies dataset.

We also included the point clouds “Man”, from Microsoft, and “Skier” from the ITI database [2]

[1] https://jpeg.org/plenodb/

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

Table I.1: 8i voxelized full bodies dataset

	Number of		
Image	**Points**	**Voxels**	**Cameras**
boxer	3496011	2056256	13
longdress	3100469	1860104	12
loot	3021497	1858707	13
redandblack	2776067	1467981	12
soldier	4007891	2365732	13

which has 229492 voxels (see Figure I.3). The other is "Objects" from the 3DCOMET dataset [64] (tagged as "r_sh_th_04"), which has 111920 voxels. All of then with a depth equal to 9. The last two ones, Skier and Objects, are available as polygon meshes and were voxelized.

(a) Man (b) Skier

Figure I.3: Rendered views for Man and Skier.