DISSERTAÇÃO DE MESTRADO

FRACTIONAL SUPER-RESOLUTION OF VOXELIZED POINT CLOUDS

Tomás Malheiros Borges

Brasília, janeiro de 2021

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

DISSERTAÇÃO DE MESTRADO

FRACTIONAL SUPER-RESOLUTION OF VOXELIZED POINT CLOUDS

Tomás Malheiros Borges

Dissertação de Mestrado submetida ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica

Banca Examinadora

Prof. Ricardo Lopes de Queiroz, Ph.D., PPGEE / UnB ______________________
Orientador

Profa. Mylène Christine Queiroz de Farias, Ph.D., PPGEE/UnB ______________________
Examinador Interno

Prof. Eduardo Antônio Barros da Silva, Ph.D., UFRJ ______________________
Examinador Externo

FICHA CATALOGRÁFICA

BORGES, TOMÁS MALHEIROS
FRACTIONAL SUPER-RESOLUTION OF VOXELIZED POINT CLOUDS [Distrito Federal] 2021.
xvi, 77 p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2021).
Dissertação de Mestrado - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. point cloud	2. super-resolution
3. quality assessment	4. compression
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

BORGES, T. M. (2021). *FRACTIONAL SUPER-RESOLUTION OF VOXELIZED POINT CLOUDS.* Dissertação de Mestrado, Publicação: PPGENE.DM-759/20, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 77 p.

CESSÃO DE DIREITOS

AUTOR: Tomás Malheiros Borges
TÍTULO: FRACTIONAL SUPER-RESOLUTION OF VOXELIZED POINT CLOUDS.
GRAU: Mestre em Engenharia Elétrica ANO: 2021

Tomás Malheiros Borges
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Q: What is the difference between a Master of Engineering and a large pizza?
A: A large pizza can feed a family of four.

Acknowledgments

Throughout the writing of this work, I have received a great deal of support and assistance.

I would first like to thank my advisor, Professor Ricardo Queiroz, whose expertise was invaluable in formulating the research questions and methodology. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

I would like to acknowledge my colleagues from the DIVP group for their collaboration. I would particularly like to thank Professor Diogo Garcia for providing me with the necessary tools for the development of this work, and Professor Tiago Fonseca for the comprehensive support, especially during the subjective evaluations.

In addition, I would like to thank my girlfriend, Rafaella, and my parents for their unconditional support. Without them, I could not have completed this thesis.

Finally, I would like to acknowledge the support of my friends, who provided stimulating discussions as well as happy distractions to rest my mind outside of my research. Particularly, I would like to thank Ronaldo and Gabriel for their kind support.

Tomás Malheiros Borges

Work partially supported by CNPq/CAPES under grant 88882.384244/2019-01.

ABSTRACT

We present a method to super-resolve voxelized point clouds downsampled by a fractional factor, using a lookup-table (LUT) constructed from self-similarities from their own downsampled neighbourhoods. Given a downsampled point cloud geometry V_d, and its corresponding fractional downsampling factor s, $1 < s \leq 2$, the proposed method determines the set of positions that may have generated V_d, and estimates which of these positions were indeed occupied (super-resolution). Assuming that the geometry of a point cloud is approximately self-similar at different scales, a LUT relating downsampled neighbourhood configurations with children occupancy configurations can be estimated by further downsampling the input point cloud, and by taking into account the irregular children distribution derived from fractional downsampling. For completeness, we also interpolate texture by averaging colors from adjacent neighbour voxels. Extensive tests over different datasets are presented, and interesting results were obtained. We further present a direct application to improve point cloud compression using MPEG's G-PCC codec.

RESUMO

Neste trabalho, apresentamos um método para super-resolver nuvens de pontos por um fator fracionário, utilizando um dicionário construído a partir de auto-similaridades presentes na versão subamostrada. Dada a geometria de uma nuvem de pontos subamostrada V_d, juntamente com o correspondente fator de subamostragem s, $1 < s \leq 2$, o método proposto determina o conjunto de pontos que podem ter gerado V_d e estima quais desses pontos, de fato, existem em V (super-resolução). Considerando que a geometria de uma nuvem de pontos é aproximadamente auto-similar em diferentes escalas de subamostragem, cria-se um dicionário relacionando a configuração de ocupação da vizinhança com a ocupação de nós-filhos. O dicionário é obtido a partir de nova subamostragem da geometria de entrada utilizando o mesmo fator s. Desta forma, leva-se em conta as irregularidades da subamostragem por fatores fracionários no desenvolvimento da super-resolução. A textura da nuvem de pontos é interpolada utilizando a média ponderada das cores de vizinhos adjacentes. Diversos conteúdos de diferentes fontes foram testados e resultados interessantes foram obtidos. Adicionalmente, apresentamos uma aplicação direta do método de super-resolução para melhorar a compressão de nuvens de pontos utilizando o codificador G-PCC do MPEG.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOTATION AND DEFINITIONS

Relevant Equations

$V = \{\mathbf{v}(k)\},\ \text{with}\ \mathbf{v}(k) = (x_k, y_k, z_k)$	Definition of the list of occupied voxels.
$V_d = \texttt{unique}(\texttt{round}(V/s))$	Definition of the grid downsampling.

Symbols

V	The geometry of a point cloud, a list of 3D positions, i.e. occupied voxels.
C	List of colors associated with the occupied voxels from V.
N	List of normal vectors associated with the occupied voxels from V.
$\mathbf{v}(k)$	Vector representing an occupied voxel from V
s	Scale factor used in the downsampling and upsampling methods.
$\mathcal{V}(k), C(k)$	The sets containing the positions and the colors of the child nodes from $\mathbf{v}(k)$, respectively.
$\varphi_M(k)$	The neighborhood configuration of an occupied voxel $\mathbf{v}(k)$. It is an (M^3-1) binary number indicating the occupancy of neighbors around an $M \times M \times M$ neighborhood. When the subscript is omitted, $M = 3$.
$\sigma(\mathbf{v}_d(k))$	The child occupancy state of parent voxel $\mathbf{v}_d(k)$. It is a $\lceil s \rceil^3$ binary number indicating which, of all possible children in $\mathcal{V}_u(k)$, are actually occupied.
ρ_φ	The density measure, calculated as the average neighborhood occupancy-rate of adjacent voxels to an occupied voxel in a 3 × 3 × 3 neighborhood.

Subscripts

d	Indicative of downsampling.
d^2	Indicative of downsampling from a previously downsampled input.
u	Indicative of NNI upsampling.
e	Indicative of upsampling by simple expansion.
sr	Indicative of the proposed super-resolution method.
LS	Indicative of usage of the Laplacian smoothing.
O	Used to indicate the original point cloud in the point-base metrics calculations.
$\mathcal{D}$	Used to indicate the distorted point cloud in the point-base metrics calculations.

Definitions

L_1 distance — Also known as city block distance, or Manhattan distance, it is the sum of lengths of the projections of the line segment between the points onto the coordinate axes. For vectors $\mathbf{a} = (a_1, a_2, a_3)$ and $\mathbf{b} = (b_1, b_2, b_3)$

$$\|\mathbf{a} - \mathbf{b}\|_1 = \sum_{i=1}^{3} |a_i - b_i|.$$

D1 metric — Point-to-point metric. In this work, we calculate the final D1 metric as the maximum between $D1_{O}$ and $D1_{\mathcal{D}}$. Where the first measurement relates to the omission of correct points in the degraded version and the second one with the excess of incorrect points.

D2 metric — Point-to-plane metric. Accounts for the perceived surface distortions. It is the D1 metric projected along the normal direction of each evaluated point.

octree — A tree data structure in which each internal node has exactly eight children.

pixel — Picture element.

predlift — The combination of the Predicting and the Lifting Transforms used in TMC13.

trisoup — Surface approximation method used in TMC13.

voxel — Volume element.

Acronyms

2D	Two-dimensions
3D	Three-dimensions
6DoF	Six Degrees of Freedom
AC	Alternating Current, represents coefficients with non-zero frequency
bpov	Bits per occupied voxel
CfP	Call for Proposals for Point Cloud Compression
CIE	International Commission on Illumination (Commission Internationale d'Éclairage)
CMY	Cyan, Magenta and Yellow color space
CMYK	Cyan, Magenta, Yellow, and Black color space
CRT	Cathode Ray Tube
DC	Direct Current, represents coefficients with zero frequency
DCM	Direct Coding Mode
EWA	Elliptical Weighted Average

fps	Frames per second
G-PCC	Geometry-based Point Cloud Compression
GPS	Global Positioning System
GPU	Graphics Processing Unit
GSM	Gaussian Scale Mixture
GSP	Graph Signal Processing
GTV	Graph Total Variation
HDTV	High Definition Television
HEVC	High Efficiency Video Encoding
HSV	Hue, Saturation, and Value color space
HVS	Human Visual System
LIDAR	Light Detection And Ranging sensor
LoD	Level of Detail
L-PCC	LIDAR Point Cloud Compression
LR	Low-resolution
LS	Laplacian Smoothing
MLS	Moving Least Squares
MPEG	Moving Pictures Expert Group
MSE	Mean-Squared Error
MSSIM	Mean SSIM
MVUB	Microsoft Voxelized Upper Body
NNI	Nearest Neighbor Interpolation
PCC	Point Cloud Compression
PCL	Point Cloud Library
PMSE	Projected MSE
PPSNR	Projected PSNR
PSNR	Peak Signal-to-Noise Ratio
PSSIM	Projected SSIM
PVIFP	Projected VIFP
QoE	Quality of Experience
RAHT	Region-Adaptive Hierarchical Transform
RBF	Radial basis function
RGB	Red, Blue and Green color space
S-PCC	Surface Point Cloud Compression
SR	Super-resolution
SSIM	Structural Similarity Index Measure
TMC13	Test Model Categories 1 and 3
TMC2	Test Model Category 2
UPM	Universidad Politécnica de Madrid
VIF	Visual Information Fidelity
VIFP	VIF in the pixel domain

V-PCC	Video-based Point Cloud Compression
VR	Virtual Reality
VVC	Versatile Video Coding
XR	Extended Reality
YCbCr	Luma, Chrominance blue, and Chrominance red color space
YUV	Luma, Chrominance blue, and Chrominance red color space, following the BT.709 HDTV standard

1 INTRODUCTION

1.1 BACKGROUND & MOTIVATION

Advancements in technologies for scanning 3D objects and scenes have increased the demand for immersive content in the past few years. A type of volumetric data acquired through these scanning sensors are called point clouds, which typically have thousands up to billions of points. Each data sample, or point, contains a 3D coordinate indicating the presence of the scanned 3D volume's hull at that location associated with some attribute, like color, reflectance, etc. A wide range of applications use point clouds. For instance, they are used in cultural heritage to preserve a digital copy of buildings or artworks and in immersive media applications, like Extended Reality (XR) and free-viewpoint video, also known as 6 Degrees of Freedom Video (6DoF). Holograms based on point clouds are even considered the new form of video in the next years [1].

Envisioning the increase of interest for immersive content and the new challenges proposed for this kind of data, the Moving Pictures Expert Group (MPEG) presented a new project in its Roadmap [2] to develop a standard for *Coded Representation for Immersive Media* (MPEG-I). In 2017, MPEG issued a Call for Proposals (CfP) for Point Cloud Compression (PCC) [3], which reinforced the scientific community interest in point clouds. More specifically, voxelized, i.e., quantized, point clouds were addressed for real-time application. In this context, MPEG has proposed two approaches for PCC. The Video-based Point Cloud Compression (V-PCC) approach converts the 3D structures into 2D projections and leverages the existent video encoders to compress point clouds. The Geometry-based Point Cloud Compression (G-PCC) approach, on the other hand, works directly in 3D using data structures like octrees and texture transforms to compress point clouds.

As the amount of data needed to represent a 3D scene with complex geometry, specularity properties, and colors, is enormous, tools and techniques that may help mitigate the storage problem and help existing codecs are desired. Super-resolution (SR), the process of taking a low-resolution (LR) version of data and increase its resolution using some interpolation method, is a well-studied field in Image Processing. However, it is still incipient in PCC since 2D techniques are not easily transferrable to 3D data. SR can be used in several applications such as improved rendering, interpolative compression [4], and context generation for compression [5]. Therefore, developing a method for point cloud SR is a subject of interest for the scientific community. However, the problem of assessing the quality of the SR method remains with unclear answers. Point cloud objective assessment metrics are still trailing, and, although the majority of the existent metrics were developed to capture compression artifacts, they do not have a strong correlation with subjective metrics [6]. Thus, not only creating a super-resolution method is challenging enough, but properly evaluating its quality is still an open problem.

1.2 PROBLEM STATEMENT

Super-resolving point clouds is not trivial, although useful. Few works were done in this field, especially considering the SR of voxelized point clouds and their color attributes. Resampling point clouds using fractional scale factors may be useful because it allows for greater control than using only integer factors. Fractional downsampling is used, for example, in the G-PCC as a means to achieve lossy geometry. However, there is no discussion about the theory of fractional resampling of voxelized point clouds in the literature. Visual assessment of SR methods is also necessary, however challenging.

1.3 OBJECTIVES

The objectives of this work are:

1. To review the literature needed to understand the fundamental challenges of working with point clouds, the current state-of-art of PCC, and the tools needed to study point cloud processing. To present a discussion about the theory and properties of fractional resampling of voxelized point clouds.
2. To propose a new SR method for voxelized point clouds and with an application in PCC.
3. To propose a framework to assess the quality SR methods over voxelized point clouds.

1.4 MANUSCRIPT PRESENTATION

In Chapter 2, the concepts required to explain the proposed method are briefly presented. Then, in Chapter 3, the methodology of the work is presented, and we proposed an SR method for voxelized point clouds using fractional scale factors. Also in this Chapter, we present a point cloud renderer application that could be used for subjective quality assessment. In Chapter 4, a framework to evaluate the method's quality is presented, along with the obtained results and discussions, and with a PCC application. Finally, in Chapter 5, conclusions and suggestions for future works are presented.

2 LITERATURE REVIEW

> *"Dear reader, if thou art bored with this wearisome method of calculation, take pity on me who had to go through with at least seventy repetitions of it, at a very great loss of time."*
>
> —Johannes Kepler, *Astronomia Nova,* 1609

2.1 POINT CLOUDS

2.1.1 Definition

Volumetric data consists of samples representing the value of some property of the data at a 3D location (x, y, z) [7]. These samples are referred to as *voxels* (volume element), and the properties associated with them are called *attributes*. Those attributes can indicate a variety of things. They can simply differentiate background from the presence of an object in space or can assume more complex roles like density, heat, pressure, color, among other properties, or a combination of them depending on the application. When volumetric data is acquired using a 3D surface scanning over complex object surfaces, a scattered 3D point set, also called a *point cloud* is obtained [8]. An example of a 3D captured object in its point cloud format is shown in Figure 2.1.

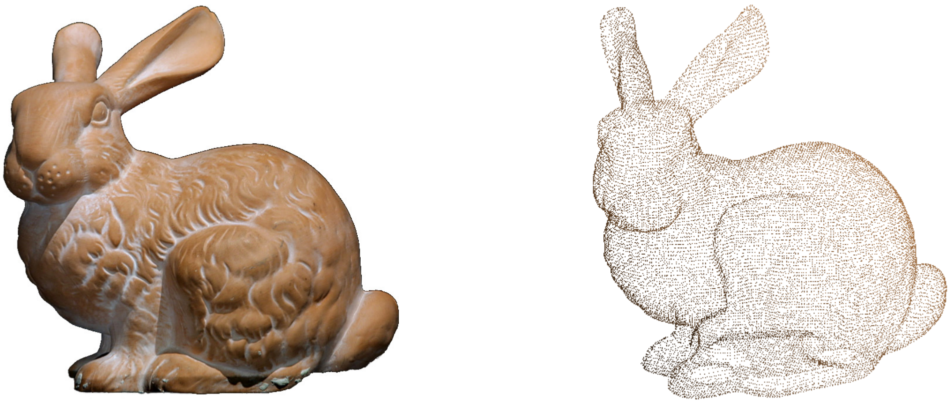


Figure 2.1: The Stanford Bunny. One of the most commonly used 3D models in computer graphics, which was captured using a range scanner [9], [10].

Point cloud data, thus, comprise a list V containing K occupied voxels. For the applications of this work, the main attribute considered is a list C of surface colors, and sometimes a list N of surface normals. The following notation is used:

$$V = \{\mathbf{v}(k)\}, \text{ with } \mathbf{v}(k) = (x_k, y_k, z_k), \tag{2.1}$$

$$C = \{\mathbf{c}(k)\}, \text{ with } \mathbf{c}(k) = (R_k, G_k, B_k), \tag{2.2}$$

$$N = \{\mathbf{n}(k)\}, \text{ with } \mathbf{n}(k) = (n_{x_k}, n_{y_k}, n_{z_k}), \tag{2.3}$$

for $k = 1, 2, \dots, K$. Notice that, although V does not need to be sorted in any particular order, the attributes C and N must be sorted in the same order as V.

For efficient processing, point clouds are usually quantized in a cubic grid. Thus, henceforth we narrow the voxel definition to a *quantized* volume element. For example, in an integer-defined 3D grid, a voxel will be any of the $1 \times 1 \times 1$ cubes forming the grid. This quantization process is known as *voxelization*, and point clouds quantized in this way are called voxelized point clouds.

2.1.2 Applications and Acquisition

Point cloud data may be generated analytically. Nevertheless, a broader range of applications is found when data is obtained by sampling real-world 3D scenes. Such point clouds can be easily acquired. They are versatile, capable of representing any kind of 3D structures (even non-manifold ones). Also, their data structure's simplicity allows for real-time processing and rendering at a relatively low computational cost. Applications include cultural heritage, 3D free-viewpoint video, real-time immersive telepresence, mobile mapping, and autonomous navigation [11], [12]. A division of the point cloud datasets was proposed by the MPEG depending on the existence of temporal information, in order to address compression strategies differently. Three categories were proposed: static, dynamic, and dynamically acquired point clouds [3], as exemplified in Figure 2.2.

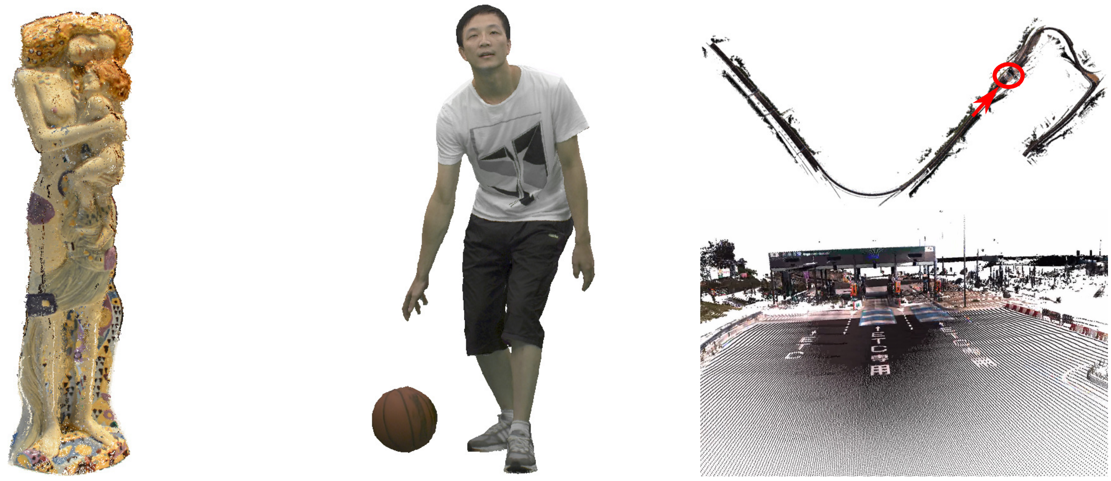
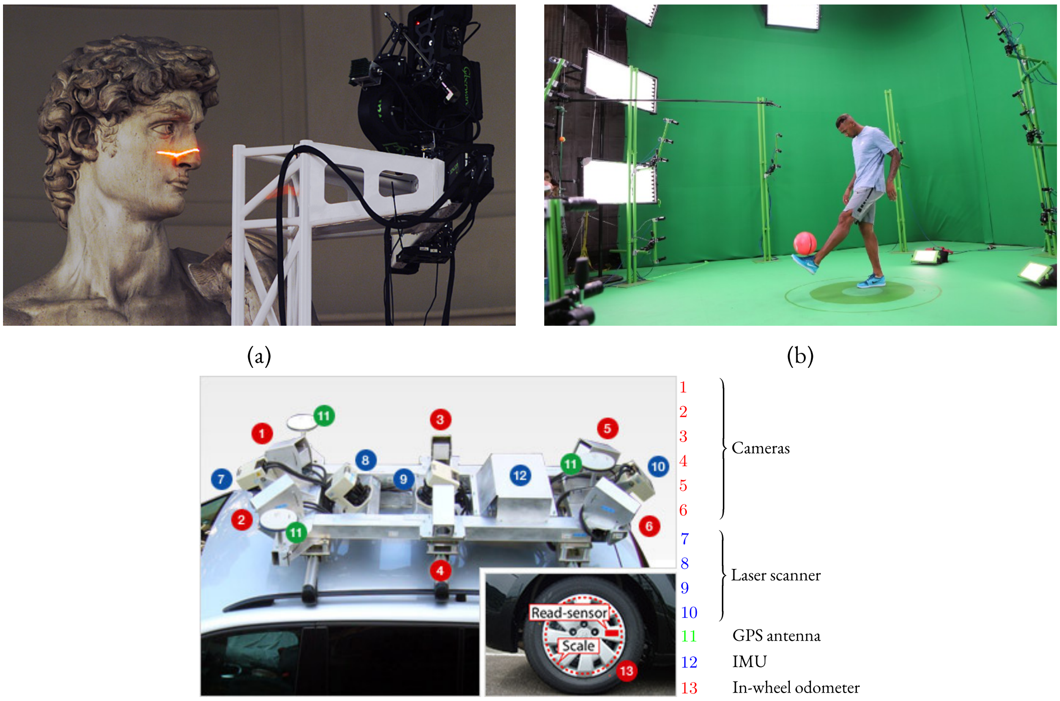

(a) (b) (c)

Figure 2.2: Examples of different point cloud contents according to MPEG's classification. (a) Category 1: Static Objects and Scenes, a cultural heritage content of *Statue_Klimt* [13]. (b) Category2: Dynamic Objects, one frame of *basketball_player* [14]. (c) Category 3: Dynamic Acquisition, *toolbooth*. Overhead view of the entire point cloud on the top, and a close-up view of the marked location on the bottom [15].

Static point clouds are datasets with no temporal variation. They are created by capturing static objects such as buildings' interiors and facades, or cultural heritage, to preserve these structures as digital copies, which can be used in museums [13], [16], [17]. Also, since scanning technologies can be very precise, with finer than 1 cm of geometric precision [12], those point clouds can capture intricate details suited for industrial assemblies and 3D printing applications [18], [19]. Dynamic point clouds comprise the data where there is movement, or temporal information, in the captured 3D volume. They are employed in real-time applications for immersive media like telepresence and virtual reality (VR) with interactive parallax [20]–[25]. For offline applications, iso-surfacing techniques may be applied to generate a polygon mesh to improve rendering quality. Polygon meshes created from point clouds are used in the replay technology of sports broadcasts to generate 3D free-viewpoint videos [26]. Lastly, dynamically acquired point clouds convey temporal information about the captured 3D scene surroundings, and it is the sensor position itself that changes with time. These are mainly used in autonomous navigation systems based on large-scale 3D dynamic maps [15], [27]–[29].


(a)

(b)

(c)

Figure 2.3: Acquisition examples. (a) Range scanner of Michelangelo's David [30]. (b) A volumetric studio [31]. (c) An example of a Mobile Mapping System (MMS) used to acquire Category 3 point clouds [27].

Static and dynamic point clouds are captured either by active, passive, or a combination of both acquisition methods. Passive acquisition methods are the ones in which the sensors do not directly give a 3D position of the sampled data. Image matching, photogrammetry, and space triangulation are required to infer the distance between object and sensor. The passive acquisition is usually made with an array of RGB cameras [32]–[35]. In studios set up to capture high-quality point clouds, called volumetric studios [23], besides passive RGB cameras, active sensors, i.e., infrared depth cameras, or range scanners, together with structured light source illumination are used. Figure 2.3(a) shows a range scanners capturing David's face, while in Figure 2.3(b) a volumetric studio is depicted. For dynamically acquired point clouds, usually light detection and ranging (LIDAR) sensors on top of vehicles are used, as shown in Figure 2.3(c). By combining azimuth and elevation information from the emitted laser beam with information about the range and the intensity from the returned reflections, one arrives at relative point locations. Adding to these relative locations, the GPS, and the inertial measurement unit (IMU) information from the vehicle, it is possible to convert the relative positions into absolute ones, tied to a geographic coordinate system [12].

2.2 VOLUME VISUALIZATION

Our depth perception comes from both monocular and binocular cues. Binocular vision allows for stereopsis, or stereo vision, which is the visual brain's ability to register a sense of 3D from visual inputs coming from both eyes [36], [37]. In stereo vision, each eye sees a slightly different version of the scene, which is projected to the retinas as 2D images, and then the brain extrapolates depth. Monocular vision perceives depth by information outside the 3D volume itself. It uses cues such as motion, perspective, lighting, shading, and depth of field [37]. Thus, to visualize volumetric data, we first need to find a way to project a 3D volume into a 2D screen. Then, to give the depth perception, we need to add some depth cues to the scene.

Volume Visualization is the field of Computer Science that targets this problem. Its main role is to create images that convey various insights about the input volumetric data [8]. Possible solutions vary depending on user/application requirements and computational restrains. Nonetheless, all such solutions follow the same conceptual framework called the volume visualization pipeline [8], [38], as depicted in Figure 2.4.

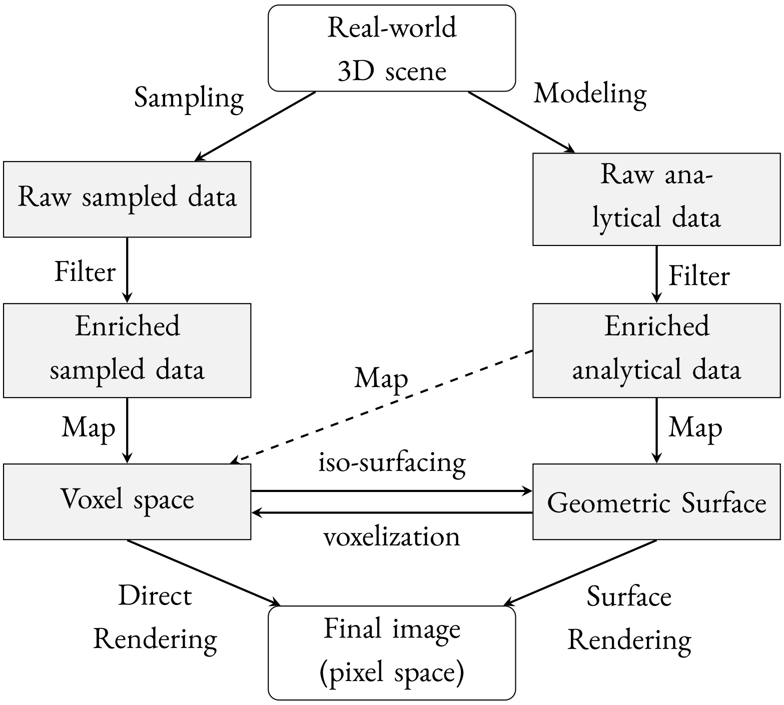


Figure 2.4: The Volume Visualization Pipeline

First, raw volumetric data is acquired. It may be by sampling 3D data, as described in Section 2.1, or by modeling analytical data. Then, there is a filter step to enhance data, where operations such as selecting only a subset of the imported dataset, outlier removal, coordinate transformation, filtering, resampling, or quantization are performed. The next step is to map abstract data to visual representation. This may be done in the voxel (discrete) space or using interpolations and connectivity information to create a geometric surface representation, i.e., polygon mesh. Transforming from voxels to geometric surface is possible using iso-surfacing algorithms. The inverse transformation is also possible by voxelizing the geometry surface in discrete and quantized points. Sometimes, although rarely, analytical data is directly mapped to the voxel space. The final process is the rendering step, in which the mapped 3D data is rendered, or "drawn" into the 2D screen, together with the user-specified viewing parameters such as viewpoint and lighting [8]. There are many rendering algorithms, and they are usually divided amongst direct and indirect methods. Direct rendering methods render the volumetric data directly from voxels to pixels, while indirect methods, like surface rendering, require an additional interpolation method. Indirect methods usually have better-perceived quality, especially when few volumetric samples are available. However, there is an added computational cost and storage. For real-time applications, direct methods can provide a faster and more straightforward, albeit less accurate, implementation [8].

In the following Sections, some aspects of point cloud visualization are discussed. The goal is to present the compromises that have to be made in order to render a point cloud. We focus on rendering strategies considering quality assessment purposes for real-time applications.

2.2.1 Virtual Camera

Before imaging the mapped volumetric data, a virtual camera must be set up in order to provide both a point of view and the necessary parameters for projecting the 3D volume in a 2D plane. A virtual camera is specified by its extrinsic and intrinsic parameters. The eye e, and center c positions, and a vector $\mathbf{u}$ indicating the camera's up axis are the extrinsic parameters, which express where the camera is, where it is pointing at, and its up direction, respectively. Intrinsic parameters specify how the camera operates, thus defining the focal length z_{near}, the field-of-view angle ϕ_{fov}, the aspect ratio w/h, and the distance of the far clipping plane z_{far}. These parameters define the view volume in a pyramid frustum shape, also called the view frustum, as depicted in Figure 2.5. Only objects inside this volume can be "seen" by the camera.

The virtual camera works similarly to a real camera by capturing a perspective projection in which all light rays coming from the 3D scene converge at a common point. The image is formed on a planar surface between the center of projection and the 3D scene. In perspective projections, objects further away from the view plane appear smaller, an effect called foreshortening. Although this is the more natural perspective configuration, sometimes foreshortening is not desirable, especially when some form of distance measurement needs to be performed in the projected image. By setting the center of projection to infinity, light rays coming from the scene become parallel, and the view volume from Figure 2.5 becomes a rectangular parallelepiped. This projection configuration is called parallel or orthographic, and it is useful when it is required for equal 3D distances to appear as equal 2D distances on the view plane [8]. Figure 2.6 depicts the differences in the projection formation, considering perspective and orthographic configurations.

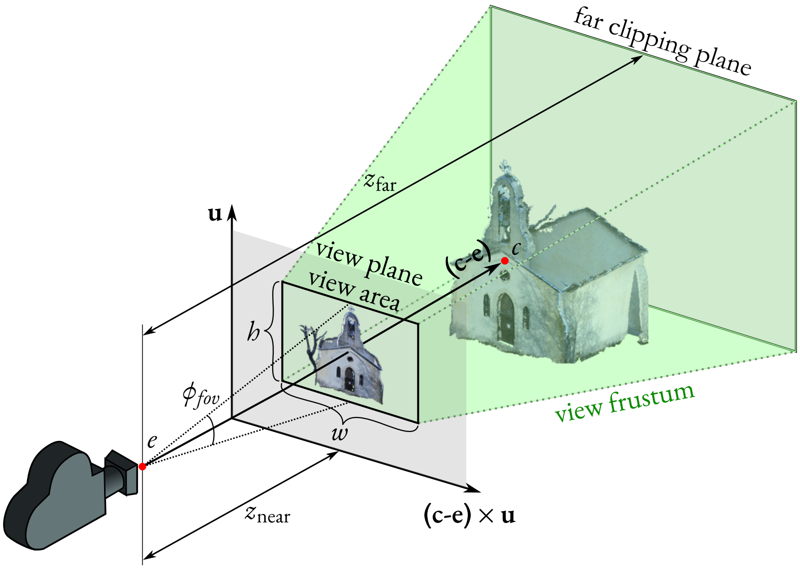
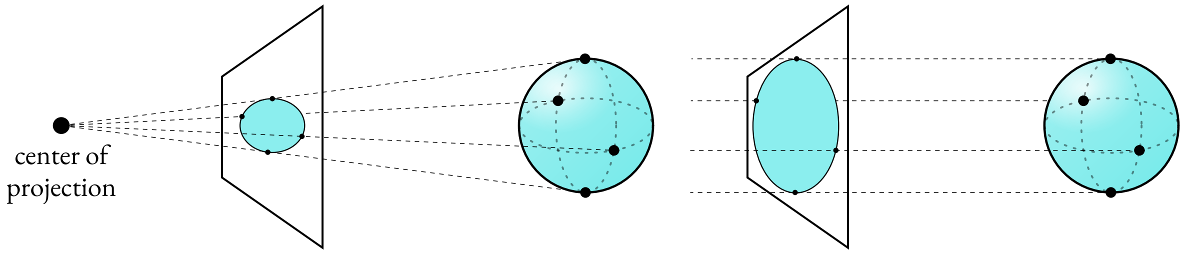


Figure 2.5: The virtual camera parameters [8].



(a) Perspective Projection.

(b) Orhtographic Projection.

Figure 2.6: The formation of 2D images considering different positions of the center of projection.

By allowing interactive changes in some of the camera parameters (eye, center, up, and field-of-view), some monocular depth perception cues are triggered, giving the user an immersive experience.

2.2.2 Color Spaces

In order to convey color for point cloud data, it is necessary to specify the color space in which this attribute was stored. Color spaces are multidimensional spaces in which each dimension represents the different components of color or channels [39]. Each space was designed either because of similarities with the human vision or because they are more suited for some specific application.

The red, green, and blue (RGB) space is based on the tri-chromatic nature of the human visual system (HVS). The RGB's primary colors were standardized by the International Commission on Illumination (CIE, for its French name, Commission Internationale d'Éclairage) using wavelengths: 700.0nm for red, 546.1nm for green, and 435.8nm for blue [40]. RGB is usually the system chosen for storage or display of color in hardware-oriented applications, as is the case for video monitors and cameras. Its origin is related directly to the three-electron-gun (one red, one green, and one blue) organization of colored cathode ray tube (CRT) displays [7]. This still holds, nowadays, as modern display subpixels and sensors are usually RGB. RGB is an additive system, which means that every color is represented as a mix of its primary colors in different amounts. Usually, components are in the range $[0, 1]$ when represented by floating-point numbers, or $[0, 255]$ when 8-bit depth integer representation is used [8]. The absence of all colors means the absence of light, thus, representing black. Equal amounts of the three colors determine gray shades, and the maximum amount of all colors determines white.

The cyan, magenta, yellow, and black (CMYK) space, differently from RGB, is a subtractive space developed for printing devices that deposit pigment in white paper. In subtractive systems, the absence of the color components represents white (the paper color), while equal amounts of CMY leads to an imperfect black. In order to work around this issue and save ink for printers, a black pigment component was added to the system.

Another popular color space that shares similarities with the human vision system is the hue, saturation, and value (HSV) color space. Hue distinguishes between different colors of different wavelengths; saturation can be seen as how much the hue is diluted with white, and value represents the brightness, or luminance, of a given color [8]. HSV is natural and approximately perceptually uniform; therefore, the quantization of this space can produce a compact and complete collection of colors [39].

Finally, there are the chrominance-based color spaces. These spaces are orthogonal and use statistical independent components. Because the eye has more spatial acuity for luminance than for color, the chrominance components can be further compressed without perceptual loss in quality, which makes these color spaces suited for image compression [40], [41].In YUV, a chrominance-based space that is used in TV standards, Y represents luma (a luminance approximation that uses gamma-corrected component in its calculation), which can be comparable to the regular black-and-white TV signal, and the lower frequency chroma channels (chrominance blue U, and chrominance red V), which convey color to the signal [40]. It is also the color space of choice commonly used in color metrics methods for point cloud assessment [3]. The conversion from RGB to YUV following the BT.709 standard for HDTV [42] is the one used whenever the YUV color space is mentioned in this work:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.2126 & 0.7152 & 0.0722 \\ -0.1146 & -0.3854 & 0.5000 \\ 0.5000 & -0.4542 & -0.0458 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 0.5 \\ 0.5. \end{bmatrix} \tag{2.4}$$

2.2.3 Point Cloud Rendering

The canonical methods for rendering point clouds are the direct ones. In contrast, surface rendering methods result from some 3D reconstruction algorithm and require prior assumptions on topology and sampling [43], as expressed in Figure 2.4. When rendering for quality assessment purposes, we usually search for methods that allow for a one-to-one voxel-pixel correspondence.

Also, methods that add little to no distortion to the original mapped data are preferred, such that rendering artifacts do not interfere with the assessment. In this context, the two main strategies utilized for point cloud rendering are point-based rendering and voxel-based rendering.

A simple and effective technique for rendering 3D scattered points representing surfaces is splat rendering. Splats are surface elements or *surfels*; the basic idea behind them is that if we can assume a surface to be almost flat in a neighborhood of radius R_k around every point $\mathbf{v}(k)$, then the surface can be interpolated in that region by a 2D radial basis function (RBF) [8]. Another way to view splatting is by projecting each 3D point in the image plane and associating the projection with a footprint function [35].

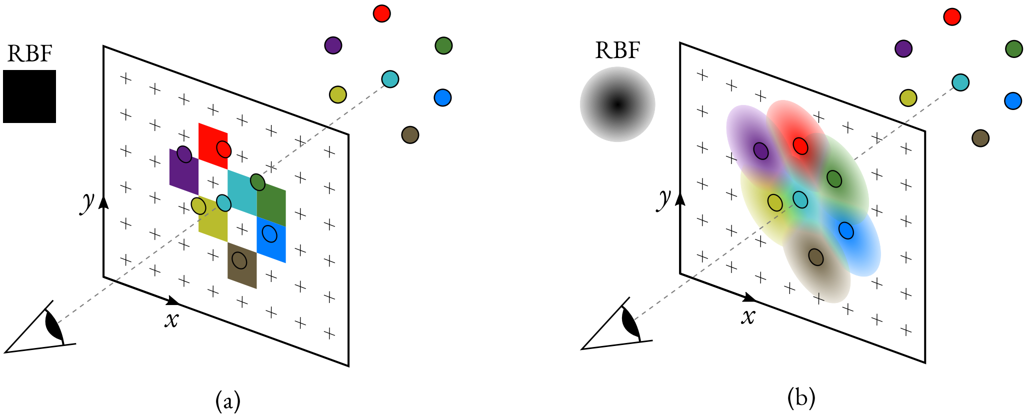


Figure 2.7: Point rendering by splatting. The naïve splatting approach in (a) compared with the EWA splatting scheme in (b) [35].

The simplest solution for splat rendering is using a constant 2D RBF, which can be implemented using point primitives. Each 3D point is projected to the image plane, assigning the color of the point to the closest pixel(s) forming the footprint function, as depicted in Figure 2.7(a). In voxelized point clouds, it is possible to align the 3D voxel grid with the 2D pixel grid to get one-to-one voxel-pixel correspondence using this rendering method.

A popular implementation of splat rendering utilizes 2D Gaussian RBF, performing an Elliptical Weighted Average (EWA) filtering [44] in the resulting 2D image, as shown in Figure 2.7(b). This approach is capable of generating high-quality images [45]. However, the added filtering step is equivalent to a resampling process [35], which may not be desirable for quality assessment purposes.

In splat rendering, each point is associated with a 2D billboard, which in computer graphics is a 2D-shaped texture object; usually, a square or circle, that always appear parallel to the view plane for better visibility [46]. The splat size is defined by the user, and typically, the point cloud's intrinsic resolution value is chosen, i.e., the mean distance of every voxel and its nearest neighbor considering real-world resolution. Most commonly, this size is fixed, representing a constant number of pixels in the projected image regardless of the zoom level, as is the case for several commercial renderers [47]–[49]. However, holes appear in close-up projection when using fixed-size splats, giving the impression of a sparser point cloud. If a larger splat size is used, the sparsity problem is solved for zoomed-in viewpoints but deforms the point cloud when the camera is set far away from the scene. Figure 2.8 illustrates the fixed-size splat rendering problem.

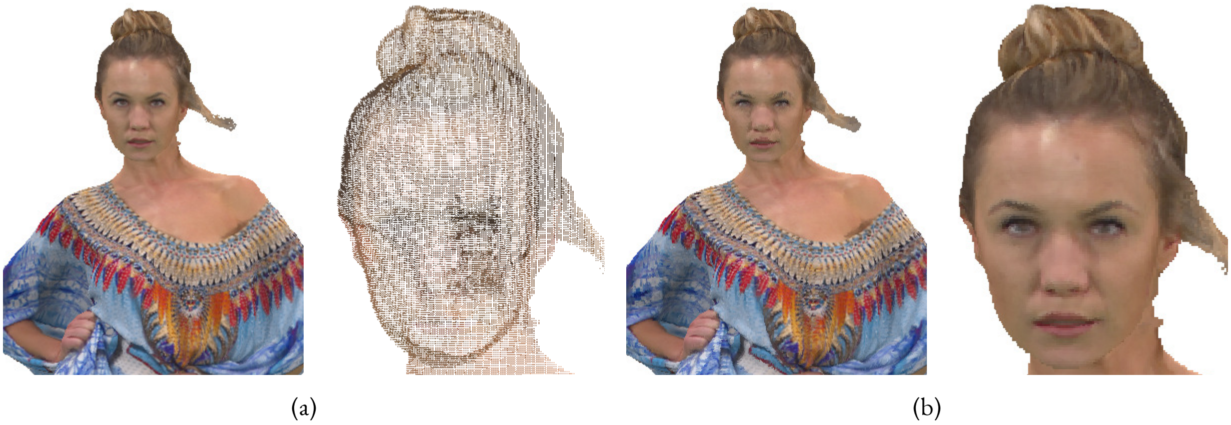


Figure 2.8: Fixed-size splat rendering of *longdress* [50]. In (a), the point size was set to generate a watertight representation for the point cloud on the left, resulting in a sparse point cloud when zoomed-in the right. In (b), the point size was defined for the zoomed-in point cloud of the right, resulting in a deformed point cloud when zoomed out in the left.

A simple solution to this problem is to automatically update the splat size every time the user performs a zoom command. One example of such a rendering solution was used in the work of Alexiou *et. al*, where a web-based renderer was created for subjective quality assessment tests [6]. Since the billboards utilized in splat rendering are always parallel to the view plane, the splats become misaligned when the camera is rotated, giving a fish scale effect on the rendered image due to the superposition of neighboring splats, as shown in Figure 2.9. This was worked around in [6] by limiting the maximum allowed zoom level, such that the end-user could not perceive this effect.

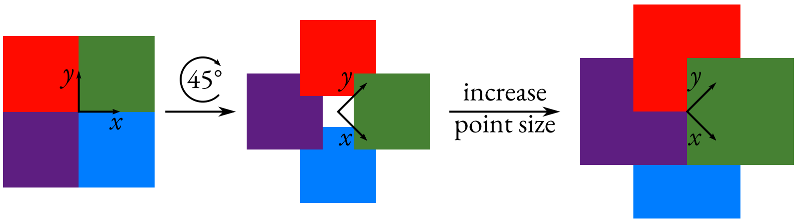
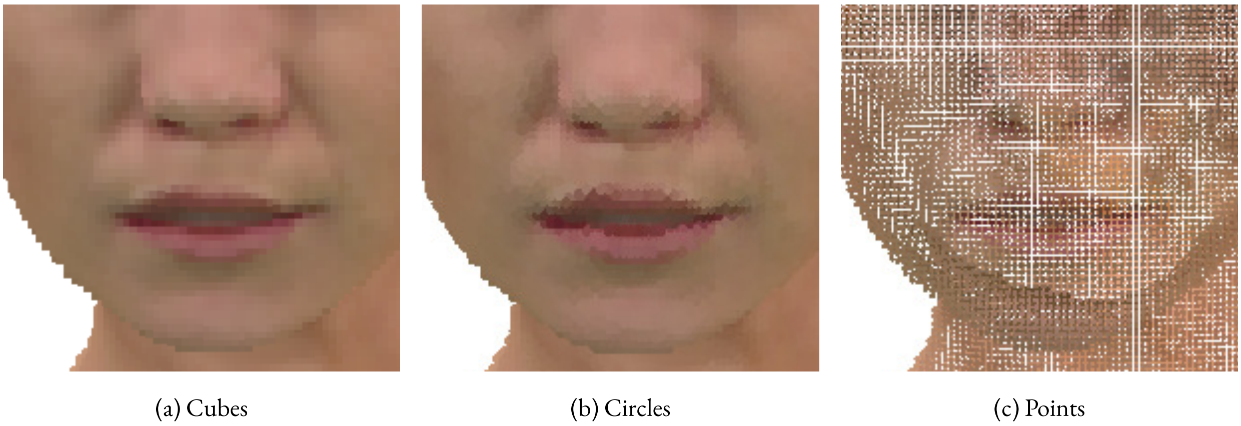


Figure 2.9: Illustration of the fish scale effect on splat rendering. Since billboards are always aligned parallel to the view plane, they get superposed when the camera is rotated. This creates the fish scale effect, and, if the point size is not big enough, it also creates holes.

The other approach for point cloud rendering is to do voxel-based rendering using 3D primitives available in geometric shaders of modern Graphics Processing Units (GPUs). This way, cubes are placed in occupied positions, and are sized by the voxel grid, allowing for rotation without fish scale effect or holes.

Voxel-based rendering, in its turn, has a somewhat limited capability to model complex geometry. Meaning that excessive upsampling of the underlying volumetric representation is required to increase geometric detail. In general, the overhead of volume samples for large data sets and



(a) Cubes

(b) Circles

(c) Points

Figure 2.10: Different rendering results for the three primitives available in the MPEG renderer [51], using the same point size for a close-up view. In (a), cubes are used as primitives. In (b), adaptive-size circles are used as primitives. In (c), fixed-sized points are used as primitives.

high-resolution establishes boundaries to their practical use [43].

A voxel-based rendered was proposed by Technicolor, and it is used as the default renderer for PCC visualization in MPEG [51]. Besides the option for voxel-based rendering, this renderer can also perform splat rendering using either adaptive or fixed-size circles primitives, as shown in Figure 2.10. Notice that for axis-aligned projections at the same zoom level, both voxel-based and naïve splatting rendering methods yield the same results with the desired one-to-one voxel-pixel correspondence.

Those renderers were developed for quality assessment tests in mind. Features that could improve the rendered image quality but change input data were left out, like scene relighting, which could cast realistic shadows but would change the original colors. Modern voxel-based renderers under development employ imaging improving features that can generate high-quality rendered images [52], [53].

This Section presents a shallow introduction to a complex subject. However, it provides crucial insights about some of the choices and trade-offs that must be done for different point cloud rendering applications. It is important to remember those compromises, especially when dealing with quality assessment.

2.3 POINT CLOUD COMPRESSION

Point cloud representation had its first appearance in 1985 [54] when it was proposed to use points as a way to display curved surfaces [35]. However, only when 3D capturing technologies became more accessible and reliable that point clouds experienced their renaissance. One of the first available and widespread devices for capturing point clouds was Microsoft's Kinect [55], which was launched in 2010. In 2011, an open project for 2D/3D image and point cloud processing library called Point Cloud Library (PCL) was launched [56], which gathered tools for processing point clouds and some initial compression techniques. Due to the ease of capturing 3D scenes brought by Kinect, some studies were published, and, in 2013, MPEG started discussing immersive telepresence applications using point clouds [57], [58]. A few years later, in 2016, MPEG collected requirements [59], and in the next year, studies for PCC were started with the CfP [3].

In the CfP, MPEG proposed to tackle the different datasets with different strategies. Datasets were divided into three categories, each with its own compression strategy: category 1 containing static point clouds was studied by the name of Surface Point Cloud Compression (S-PCC); category 2, with dynamic point clouds, was assigned to Video-based Point Cloud Compression (V-PCC); and category 3, comprising of dynamically acquired point clouds, was assigned to LIDAR Point Cloud Compression (L-PCC). After noticing similarities between the compression approaches used in categories 1 and 2, S-PCC and L-PCC were merged into the Geometry-based Point Cloud Compression (G-PCC) [12], [60].

Later in 2017, MPEG issued two reference software to benchmark performance and to compare new methods and proposals. Test Model Category 2 (TMC2) [61] and Test Model Categories 1 and 3 (TMC13) [62], were developed for V-PCC and G-PCC, respectively. Those models have seen significant advancements in the last years, and the PCC standard is due to be ready in late 2020, according to MPEG's schedule [2].

In the next Sections, the functioning of V-PCC and G-PCC is briefly explained.

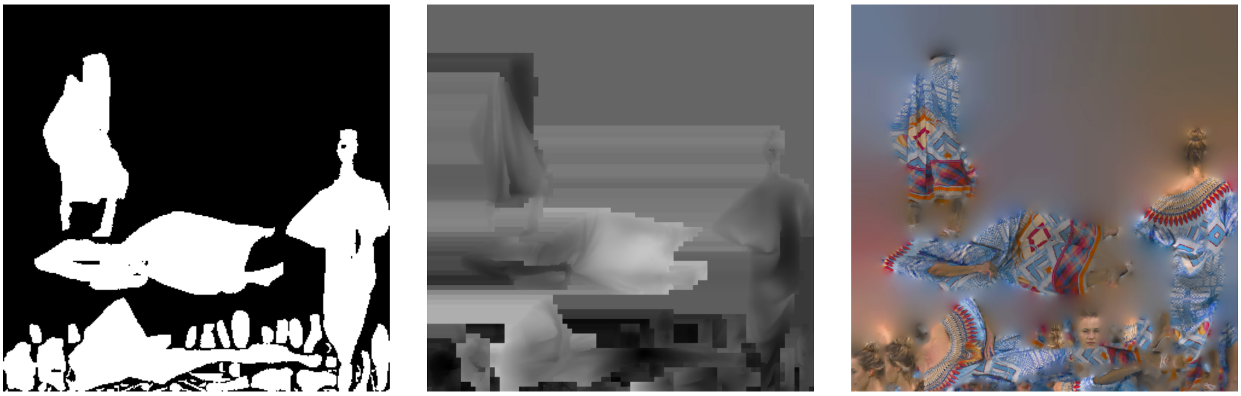
2.3.1 Video-based Point Cloud Compression

The compression of 2D videos is a mature and well-developed field. For instance, the High-Efficiency Video Encoding (HEVC) standard has impressive compression rates targeted at 300–1000:1 [63], and its successor the Versatile Video Coding (VVC) standard should have 30%–50% better compression rates for the same perceptual quality [64]. Therefore, if 3D dynamic volumes could be transformed into a 2D video, one could use the high performance and widespread availability of video codecs to encode point clouds. The proposed solution introduced by V-PCC for this problem is to divide the point cloud into 3D surface segments, decompose and project these segments into a set of 2D frames, then compress all the generated data using a standard image/video coder.

Coding Principles

V-PCC's biggest challenge is to perform compression-efficient 2D projections that allow for 3D reconstruction. The idea is to use connected 3D surface regions, called 3D patches, which are independently projected and further combined into a mosaic-like image. Independently projected regions reduce projection issues, such as self-occlusions and hidden surfaces [60]. The 3D patches are created based on the geometry of the input point cloud frame. Projections formed by different point cloud components are required to guarantee a proper reconstruction of the 3D data from the

2D video at the decoder side. Thus, the projected 2D patches are decomposed into an occupancy map (a binary image indicating which parts of the mosaic-like image should be used), far and near components for geometry (to indicate depth), and the corresponding attributes components [61], as depicted in Figure 2.11. Since the creation and positioning of patches is not straightforward, a set of instructions and parameters are associated with each patch is also required. This auxiliary information is called the atlas metadata. All the generated data is processed for highly efficient video compression. Finally, the different bitstreams are multiplexed and then sent to the video encoder.



(a) Occupancy map

(b) Geometry

(a) Attribute

Figure 2.11: Point cloud frame components used in V-PCC.

Codec Architecture

In Figure 2.12 the encoding/decoding schemes from TMC2 are presented. In order to create the 3D patches, first, each point's normal vector is estimated [65]. Then, normal vectors are quantized in the six orthographic projection directions $(\pm x, \pm y, \pm z)$. Assuming a smooth local surface, this quantization process is further refined using neighboring normal directions. Points are classified by their normal (or projection) direction, and finally clustered following their classification and using a connected components algorithm [61].

Since the projections are orthogonal and axis-aligned, two of the three coordinates of each voxel are preserved in their pixel projection. The third coordinate is determined using the geometry images, containing depth information of each voxel. To avoid multiple points projected in the same pixel, the use of far and near geometry images, among other strategies, are employed [61].

To create the mosaic-like image of Figure 2.11, patches are combined in an iterative process called packing, which aims to fit, in a Tetris-like manner, all the independent 2D patches in a $W \times H$ image. TMC2 seeks matches between patches of different frames and tries to insert matched patches at similar locations to generate a temporally consistent packing. It, then, fills in the unused spaces (image padding) of the geometry and of the attribute images for higher compression efficiency.

Outside the encoding loop, the reconstructed geometry may optionally be smoothed as a post-processing step. Since the geometry reconstruction might not be perfect, a recoloring step is per-

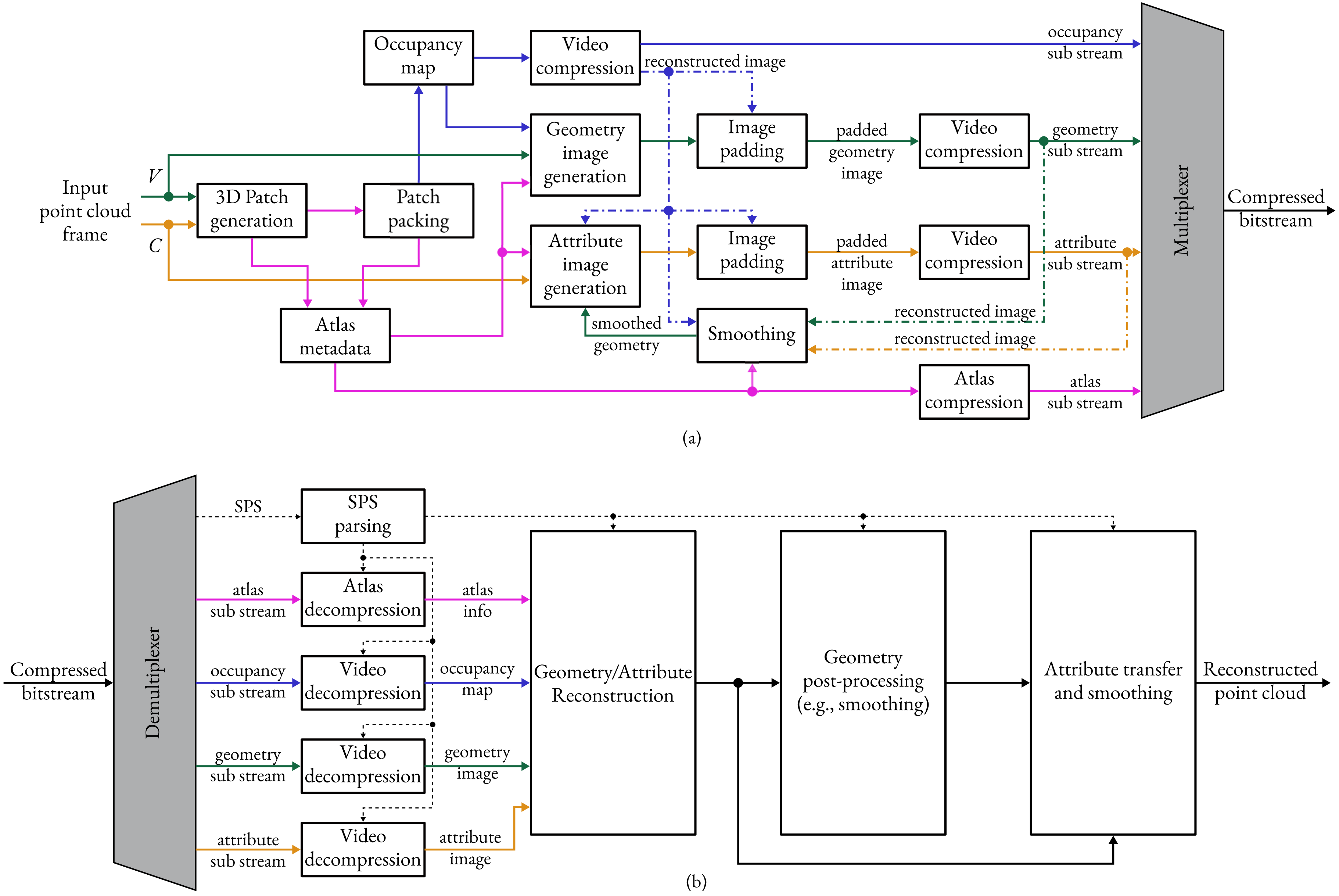


Figure 2.12: TMC2 encoding (a) and decoding (b) schemes [61].

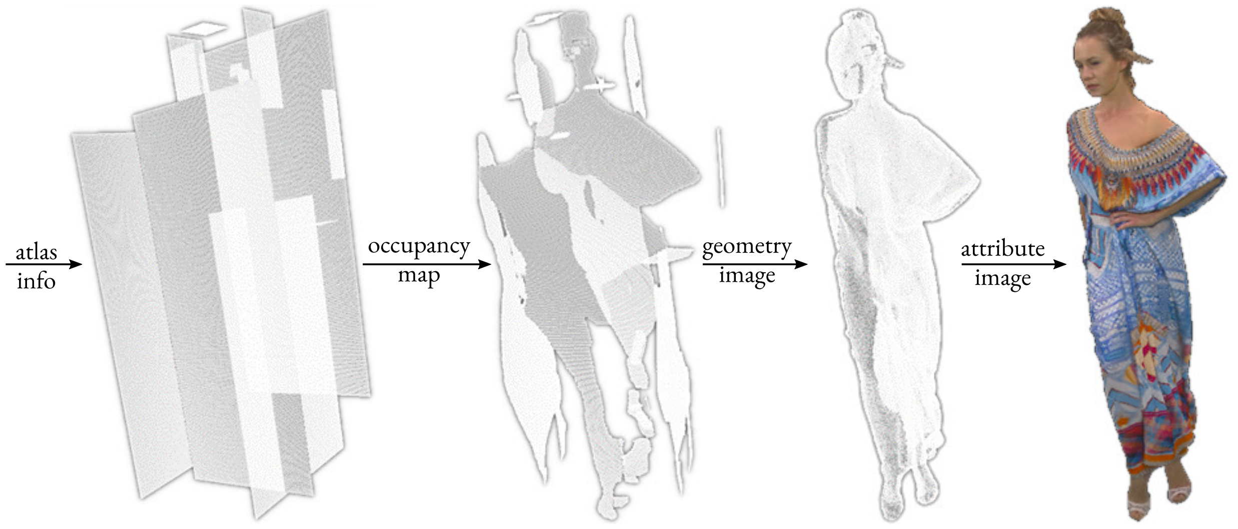


Figure 2.13: Point cloud reconstruction pipeline from TMC2 [61].

formed to adequate the color attribute. The atlas information stream is entropy coded. Finally, all the separate bitstreams are multiplexed into the output compressed binary file at the end of the process.

At the decoder side, the input compressed binary file is demultiplexed into geometry, attribute, occupancy map and auxiliary information, which are then used to reconstruct the point cloud following the pipeline showed in Figure 2.13

2.3.2 Geometry-based Point Cloud Compression

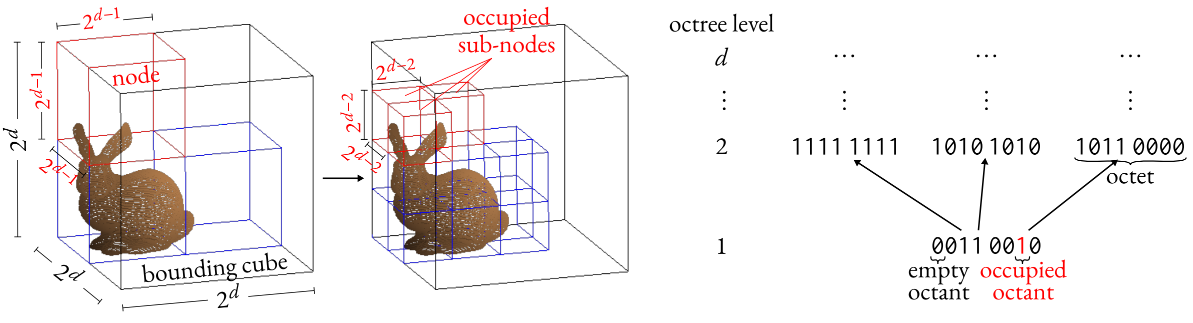
Differently from V-PCC, the G-PCC approach for compression of point clouds is directly done in 3D. By avoiding 2D projections, which require a certain density for the patch segmentation to work, this approach broadens the range of targeted input point clouds datasets. Since G-PCC was originated from the union of L-PCC and S-PCC, it inherited different configurations that the user can choose from to better suit the coder to the input data.

Coding Principles

In G-PCC, datasets are expected to have complex geometry and often exhibit irregular point sampling. As a result, geometry is first processed and compressed, followed by attribute compression. Geometry is represented in an efficient data structure manner, then entropy encoded. Attributes are subband decomposed using wavelet-based transforms, then quantized, and entropy encoded. The two bitstreams constitute the compressed point cloud. As G-PCC was planned to deal with vast point sets, functionalities for partially encoding/decoding and support for parallelized coding are required. G-PCC was originally expected to be used with datasets without temporal information. At the current point of development, it does not include any tools for temporal prediction; only intra-frame prediction is used.

The encoding/decoding scheme used in TMC13 is depicted in Figure 2.14. Since G-PCC is agnostic to the input data coordinate representation, there is a coordinate transformation, followed by a voxelization pre-processing step before the geometry coding. Input coordinates are transformed such that all points of the input data lie in a bounding cube $[0, 2^d)^3$, for some non-negative integer parameter d. Voxels are then represented by coordinates representing the center of any of the unit cubes $[i-0.5, i+0.5) \times [j-0.5, j+0.5) \times [k-0.5, k+0.5)$, for i, j, k integers between 0 and $2^d - 1$ [62]. All points from the original set that lie within a voxel's boundary are mapped to that voxel. This mapping step may result in multiple points with the same position—duplicate points. Usually, duplicate points are consolidated into one, and their attributes are averaged.

This pre-processing step conforms data into TMC13 format, allowing for efficient 3D representation. Usually, only about 1% of the voxels of the bounding cube are occupied. The octree [66], a hierarchical-tree data structure that has been shown effective to code sparse 3D objects [67], [68], is the data structure chosen for representing 3D points in TMC13. After the coordinate transformation and voxelization processes, building the octree is straightforward, as illustrated in Figure 2.15.



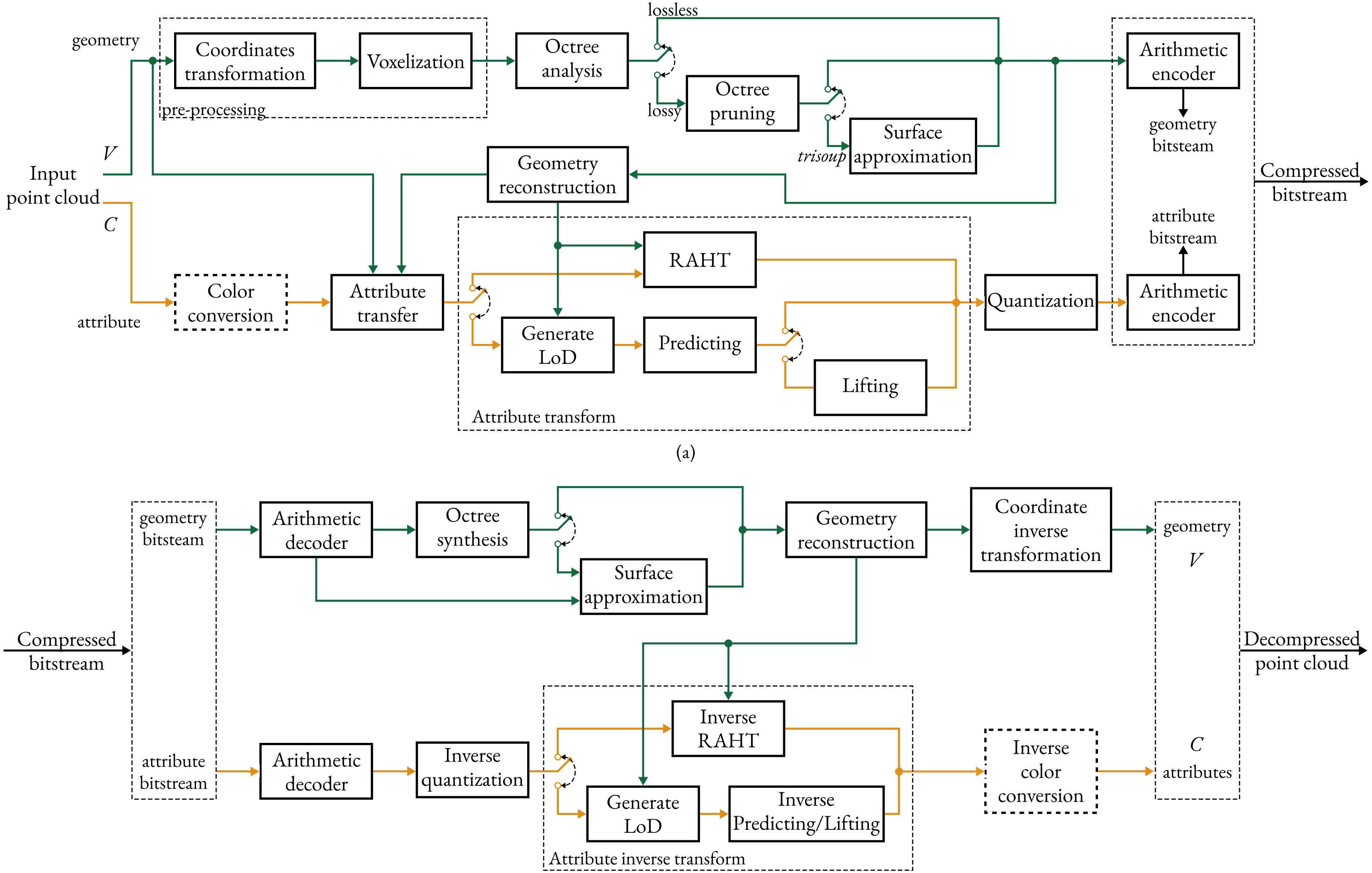
(a)

(b)

Figure 2.15: Octree analysis illustration. Graphic representation(a), and the correspondent tree representation (b).

The root node is the $2^d \times 2^d \times 2^d$ bounding cube, which is divided into eight sub-volumes, or octants, with 1/8 of the volume of its parent. Those octants containing points are marked as 1 and further divided; otherwise, they are marked as `0`, and the division stops. At each node division, eight child nodes are generated, represented by a 1-byte word, an octet, in the tree. Thus, each octree level refines the coordinates of the points within a sub-volume by using one bit for each component, at a total cost of eight bits per refinement [62]. The subdivision of occupied nodes is recursively repeated until the octants reach the smallest volume element, a $1 \times 1 \times 1$ cubic voxel. The maximum possible number of subdivision levels is defined by the parameter d, the octree depth, and it is related to the point cloud's sampling resolution.

For lossless geometry compression, all the octets of the tree are used. In dense areas of the tree, octets are entropy coded considering the correlation with neighboring octets [60], while isolated points are coded using the Direct Coding Mode (DCM) [69]. For lossy geometry, currently,



(a)

(b)

Figure 2.14: TMC13 encoding (a) and decoding (b) schemes [62].

TMC13 offers two possibilities: either by just pruning the octree from the root to an arbitrary level or by combining the octree pruning with a surface approximation. A surface reconstruction approximation from the entire octree is made using a series of triangles in a mesh-like structure, but without the connectivity information relating the triangles, this method is called *trisoup* (from triangle soup) [70]. The octree is pruned at a user-defined level, then the reconstructed geometry from *trisoup* consists of refining the pruned octree with the intersections of the mesh surface and the quantization grid, thus maintaining a voxelized geometry. When *trisoup* is enabled, a mixture of octree, segment indicator, and vertex position information are sent into the geometry bitstream [60].

An attribute transfer procedure (or recoloring) to determine the attribute values for the newly reconstructed geometry is done prior to the attribute encoding. Currently in TMC13, attributes can be coded using one of the three available transforms: Region-Adaptive Hierarchical Transform (RAHT) [67], Predicting Transform [71], and Lifting Transform [72].

- **RAHT**: The RAHT is a hierarchical orthogonal subband transform [73]. It is a variation of the Haar transform, which uses adaptive weights to consider different regions with empty or occupied voxels; in order words, it takes the data's sparsity into account.

  To implement the RAHT, one must follow the backward order of the octree scan (from leaves to root), combining voxels into larger cubes until reaching the bounding cube. Occupied voxels are assigned with weight $w = 1$, and empty ones with $w = 0$. Same branch voxels have their attributes combined through a linear transformation (Eq. 6, [67]), which roughly takes the weighted average and the weighted average difference of each voxel pair, generating one low- and one high-pass transformed coefficients, respectively. High-pass coefficients are ready to be quantized and encoded. On the other hand, the low-pass ones are promoted to the next level, and their weights are updated to the sum of the generating voxels weights. Further recombination with other low-pass coefficients is performed hierarchically and recursively. When occupied voxels are paired with empty ones, their attributes are directly promoted to the next level, but their weights do not change. Thus, densely populated regions get more importance than sparser ones in the transform. After reaching the root, the DC and all high

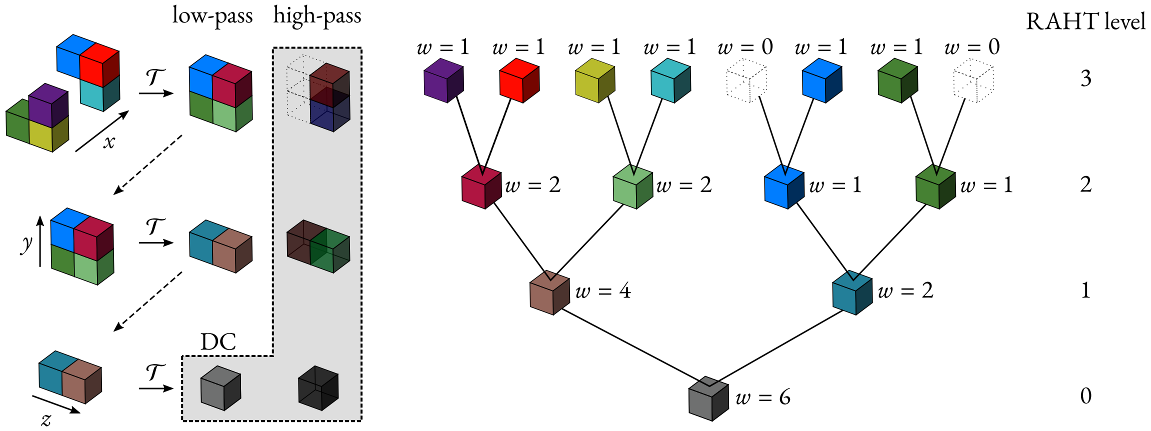


Figure 2.16: Illustration of RAHT in a $2 \times 2 \times 2$ block [60].

pass coefficients are quantized, and entropy coded [74]. Figure 2.16 illustrates the RAHT implementation for a 2 × 2 × 2 block. Notice that differently from the octree, at each level, the RAHT is performed in only one dimension; thus, one level of the octree corresponds to three levels of the RAHT.

TMC13 allows for a transform domain prediction of RAHT [75], [76]. In this approach, the RAHT octree traversal order is reversed (starting from the root and going to the leaves). Then, a step of attribute inter-depth upsampling is introduced to obtain a local prediction. This prediction is applied to AC coefficients so that only DC and prediction residual coefficients need to be encoded. The prediction of attributes is made by a weighted average of neighboring nodes. This prediction formulation of RAHT reported gains up around to 30% over RAHT without prediction [60].

- **Predicting & Lifting Transforms**: The Predicting and the Lifting Transforms follow a lifting scheme for subband decomposition [77]. In fact, the Lifting Transform is built on top of the Predicting Transform, such that both transforms are commonly referred to as *predlift*. The difference is that when using only the Predicting Transform, there is no update operator, which is commonly used for reducing the aliasing effects of the prediction step. The *predlift* forward and inverse scheme is depicted in Figure 2.17.

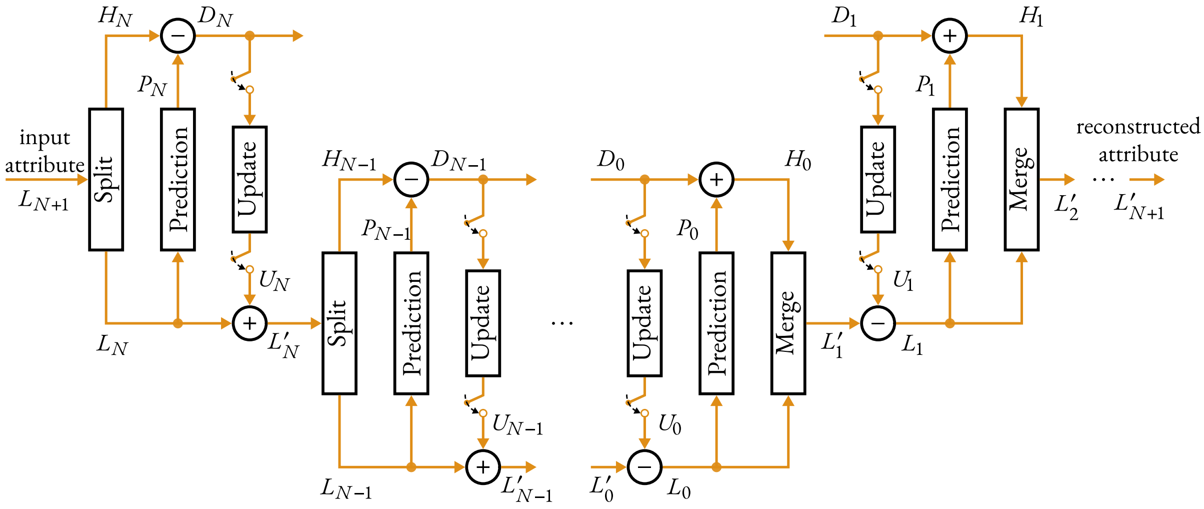


Figure 2.17: Forward and inverse *predlift* scheme [72].

The *predlift* transform relies on a Level of Details (LoD) representation, that distributes the input points into sets of refinements (R), which are created according to a set of L_1 (Manhattan) distances specified by the user [60], [62], such that $\text{LoD}_\ell = \bigcup_{i=0}^{\ell} R_i$, as depicted in Figure 2.18

Let L_ℓ be the set of attributes associated with LoD_ℓ, and H_ℓ the set of attributes associated with R_ℓ. Thus, by the definition of LoD, $L_\ell = \bigcup_{i=0}^{\ell} H_i$. The split operator takes L_ℓ as an input and generates a low-resolution output $L_{\ell-1}$, and a high-resolution output $H_{\ell-1}$. The merge operator does the inverse, taking L_ℓ and H_ℓ as inputs, and returning $L_{\ell+1}$. The prediction operator predicts the high-resolution attributes of the points in R_ℓ, by using inter-

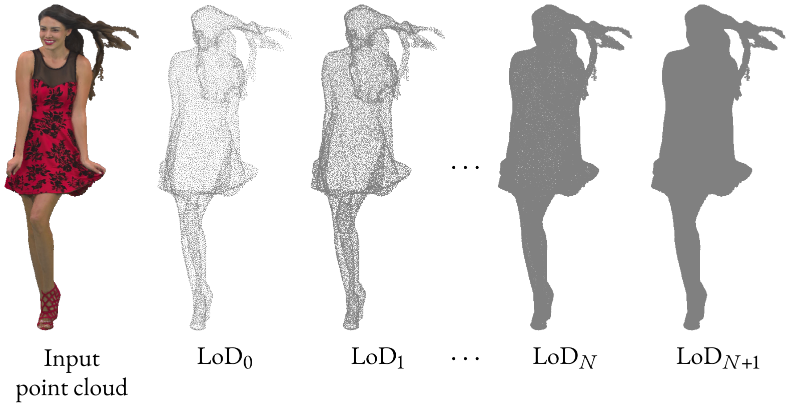


Figure 2.18: Evolution of the LoDs [72].

polation based on the inverse-distance weighted average of the k-nearest neighbors of R_ℓ in $\text{LoD}_{\ell-1}$ [72]. Finally, when the Lifting Transform is used, an update operator and adaptive quantization are introduced. Each point is associated with an influence weight based on its impact in the encoding process (points in lower LoDs have higher weights). The value of the low-resolution attributes is then updated using the prediction residuals, the distance between the predicted points and their neighbors, and their corresponding weights. Adaptive quantization is achieved by multiplying transformed coefficients by the square root of their respective weights.

2.4 POINT CLOUD QUALITY ASSESSMENT

When dealing with lossy compression, it is necessary to measure the reconstructed point cloud's visual fidelity. Quality assessment metrics allow for monitoring and improving the Quality of Experience (QoE) offered to users, which, in turn, guides the design and optimization of compression codecs and processing tools. Metrics are divided into subjective and objective, based on how the assessment is made. Subjective metrics are appraised for setting the ground truth about the perceived quality of a degraded model since they directly reflect the end-users' opinion. Nevertheless, subjective testing is highly time and money-consuming. Objective metrics, on the other hand, are designed to provide a quick and cheap prediction of subjective metrics. Usually, some form of distance (geometric or statistical) measure between the original and distorted content is employed to try to get a highly correlated prediction of the perceived subjective visual fidelity.

As discussed in Section 2.2, point clouds cannot be directly visualized; they must be rendered first, which implies that choices about the rendering algorithm must be made. This makes point cloud quality assessment especially challenging. Even the considered "ground truth" method is subjected to open questions like whether the renderer should barely change data or adapt it to reflect the intended application better. Objective metrics get even more challenging since most of the available metrics are not highly correlated with subjective scores [6]. Moreover, this correlation may vary depending on how the subjective assessment was made [78]. For these reasons, in this work we chose to focus on the point-based objective metrics, which are used in the standard tests of MPEG, available through the `dmetric`[1] software (version 0.13.4) [79]. In addition to the MPEG standard metrics, we will also consider some projection-based metrics [80], which relate to the widespread image quality assessment metrics, a more explored field that can bring good insights to QoE.

2.4.1 Point-based Metrics

The objective metrics used in MPEG's PCC evaluation procedure are full-reference quality metrics, which means that the original point cloud is used in its entirety in the comparison. They are computed symmetrically, first considering the original point cloud O as the reference, and then considering the degraded version $\mathcal{D}$ as the reference. The metrics are calculated by some form of distance between every point in $\mathcal{D}$ and its nearest neighbor in O, considering the individual degradation contribution of each point and averaging them to arrive at an overall metric.

The way the error is calculated defines the name of the metric. Errors between points define the point-to-point metric, point-to-plane uses the projected error between points and the reference's normal directions, and errors between colors define the end-to-end color metrics. Usually, Euclidean distance is used, but the Hausdorff distance is also available. The Hausdorff distance measures how far two subsets of a metric space are from each other. In other words, it is the greatest of all distances from a point in one set to the closest point in the other set [81].

For all point-based metrics, we need to find, for each query voxel $\mathbf{v}_O(k)$ in V_O of the original point cloud O, the correspondent nearest neighbor $\mathbf{v}_\mathcal{D}(k_O)$ in $V_\mathcal{D}$ of the degraded point cloud $\mathcal{D}$,

$$\mathbf{v}_\mathcal{D}(k_O) = \underset{\forall \mathbf{v}_O(k) \in V_O}{\arg\min} \|\mathbf{v}_O(k) - \mathbf{v}_\mathcal{D}(i)\|, \tag{2.5}$$

and symmetrically,

$$\mathbf{v}_O(k_\mathcal{D}) = \underset{\forall \mathbf{v}_\mathcal{D}(k) \in V_\mathcal{D}}{\arg\min} \|\mathbf{v}_\mathcal{D}(k) - \mathbf{v}_O(i)\|, \tag{2.6}$$

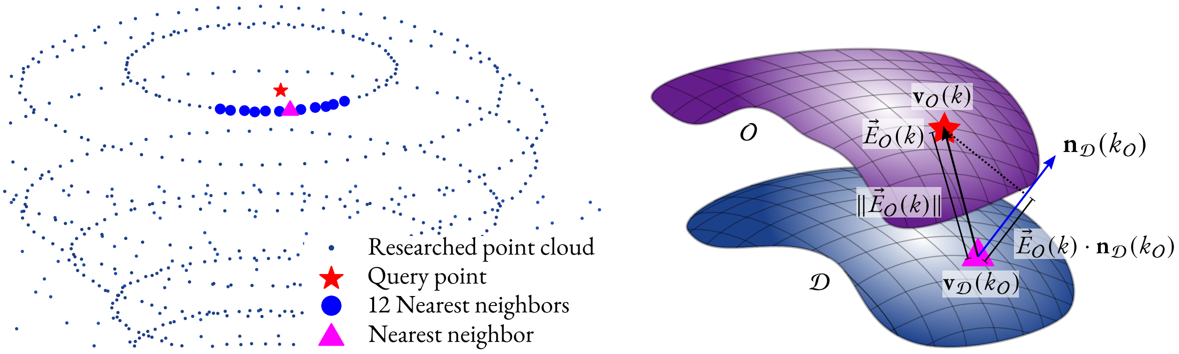
which is performed using a nearest neighbor search algorithm. Figure 2.19(a) illustrates the nearest neighbor search.

Since no attribute is evaluated in the point-to-point metric, neighboring points within the same distance are discarded, and only the first found neighbor is used. However, when considering attributes, as is the case of point-to-plane and point-based color metrics, neighboring points within the same distance need to be included, as their attribute values may differ.

2.4.1.1 Point-to-point metric (D1)

Point-to-point metrics were first introduced in order to measure data acquired by LIDAR scanners [82]. It is a simple and fast method, and it does not require any additional information but both point clouds geometries. In the `dmetric` software, it is referred to as the D1 metric.

[1] http://mpegx.int-evry.fr/software/MPEG/PCC/mpeg-pcc-dmetric.git.


Researched point cloud
Query point
12 Nearest neighbors
Nearest neighbor
$\mathbf{v}_{\mathcal{O}}(k)$
$\mathcal{O}$
$\vec{E}_{\mathcal{O}}(k)$
$\mathbf{n}_{\mathcal{D}}(k_{\mathcal{O}})$
$\|\vec{E}_{\mathcal{O}}(k)\|$
$\vec{E}_{\mathcal{O}}(k) \cdot \mathbf{n}_{\mathcal{D}}(k_{\mathcal{O}})$
$\mathcal{D}$
$\mathbf{v}_{\mathcal{D}}(k_{\mathcal{O}})$

(a) (b)

Figure 2.19: Point-based metrics illustration. In (a), the nearest neighbor search was performed to find the 12 nearest neighbors of the query point depicted with a red star in the researched point cloud. The first nearest neighbor is identified with a magenta triangle. In (b), the point-to-point and the point-to-plane errors between the query point and its nearest neighbor are graphically represented.

Using $\mathcal{O}$ as reference, we compute the error vectors $\vec{E}_{\mathcal{O}}(k)$ between every point in $V_{\mathcal{O}}$ and its correspondent nearest neighbor in $V_{\mathcal{D}}$, which were found using (2.5),

$$\vec{E}_{\mathcal{O}}(k) = \mathbf{v}_{\mathcal{O}}(k) - \mathbf{v}_{\mathcal{D}}(k_{\mathcal{O}}). \tag{2.7}$$

To compute the mean-squared error (MSE), we average the squared norm of each error vector for all $K_{\mathcal{O}}$ points,

$$\mathrm{D1}_{\mathcal{O}}^{\mathrm{MSE}} = \frac{1}{K_{\mathcal{O}}} \sum_{k=1}^{K_{\mathcal{O}}} \|\vec{E}_{\mathcal{O}}(k)\|^2. \tag{2.8}$$

Figure 2.19(b), illustrates the graphic representation of the error of (2.7).

The error calculated by (2.8) estimates the degradation of $\mathcal{D}$ using just the subset of voxels $\{\mathbf{v}_{\mathcal{D}}(k_{\mathcal{O}})\}$, the nearest neighbors of $\mathcal{O}$ in $\mathcal{D}$. When $\mathrm{D1}_{\mathcal{O}}^{\mathrm{MSE}} = 0$, it means that $\mathcal{D}$ contains all the points of $\mathcal{O}$, in other words $V_{\mathcal{D}} \supset V_{\mathcal{O}}$. Notice, however, that Equation (2.8) does not necessarily consider all the elements in $V_{\mathcal{D}}$. We need to take the symmetrical error using the point cloud $\mathcal{D}$ as reference to consider its entirety of points,

$$\vec{E}_{\mathcal{D}}(k) = \mathbf{v}_{\mathcal{D}}(k) - \mathbf{v}_{\mathcal{O}}(k_{\mathcal{D}}), \tag{2.9}$$

$$\mathrm{D1}_{\mathcal{D}}^{\mathrm{MSE}} = \frac{1}{K_{\mathcal{D}}} \sum_{k=1}^{K_{\mathcal{D}}} \|\vec{E}_{\mathcal{D}}(k)\|^2. \tag{2.10}$$

In turn, when $\mathrm{D1}_{\mathcal{D}}^{\mathrm{MSE}} = 0$, it means that all the points of $\mathcal{D}$ are contained in $\mathcal{O}$, or $V_{\mathcal{D}} \subset V_{\mathcal{O}}$. Hence, (2.8) and (2.10) are complementary. In order to arrive at the final MSE measure, either the maximum or the average of the two errors must be calculated [82]–[84]. At the current version of the `dmetric` software, the maximum error is used, such that,

$$\mathrm{D1}^{\mathrm{MSE}} = \max\left(\mathrm{D1}_{\mathcal{O}}^{\mathrm{MSE}}, \mathrm{D1}_{\mathcal{D}}^{\mathrm{MSE}}\right). \tag{2.11}$$

2.4.1.2 Point-to-plane metric (D2)

Although the D1 metric captures the differences between the points, it fails to account that the points in a point cloud represent a surface. For that matter, the point-to-plane metric was included in `dmetric` and is referred to as the D2 metric. Using O as reference, the metric is calculated by projecting, i.e., taking the dot product, of each error vector $\vec{E}_O(k)$ along the normal direction of its nearest neighbor in $\mathcal{D}$, $\mathbf{n}_{\mathcal{D}}(k_O)$, as illustrated in Figure 2.19(b). Symmetrically, when $\mathcal{D}$ is the reference, we project each $\vec{E}_{\mathcal{D}}(k)$ along the normal direction of its nearest neighbor in O, $\mathbf{n}_O(k_{\mathcal{D}})$. This way, larger penalties are imposed on errors that move further away from the local plane surface [84], which creates a perceived rugged surface; and smaller penalties are imposed for errors that move perpendicular to the local plane, which does not change the perceived surface smoothness.

The D2 metric is evaluated as follows:

$$\mathrm{D2}^{\mathrm{MSE}} = \max\left\{\frac{1}{K_O}\sum_{k=1}^{K_O}\left(\vec{E}_O(k)\cdot\mathbf{n}_{\mathcal{D}}(k_O)\right)^2,\ \frac{1}{K_{\mathcal{D}}}\sum_{k=1}^{K_{\mathcal{D}}}\left(\vec{E}_{\mathcal{D}}(k)\cdot\mathbf{n}_O(k_{\mathcal{D}})\right)^2\right\}. \tag{2.12}$$

When normal vectors $N_O = \{\mathbf{n}_O(k)\}$ are not available in O, they need to be estimated for the D2 calculations. That is done by assuming that the point cloud surface can be locally modeled by a plane [65]. Estimation of the normal vectors $N_{\mathcal{D}} = \{\mathbf{n}_{\mathcal{D}}(k)\}$ in $\mathcal{D}$ is not carried out the same way, however, since they are likely to be biased due to geometric distortions. Instead, normals from O are transferred to the voxels in $\mathcal{D}$ using their nearest neighbors [84].

The peak signal-to-noise ratios (PSNR) in dB for both D1 and D2 metrics is given by:

$$\mathrm{PSNR}_{\mathrm{D1}} = 10\log_{10}\left(\frac{p_g^2}{\mathrm{D1}^{\mathrm{MSE}}}\right). \tag{2.13}$$

$$\mathrm{PSNR}_{\mathrm{D2}} = 10\log_{10}\left(\frac{p_g^2}{\mathrm{D2}^{\mathrm{MSE}}}\right). \tag{2.14}$$

For the point clouds defined in the CTC dataset, the peak value p_g is a constant defined in Table 2 of [85]. As a rule of thumb, p_g is the length of the bounding cube's diagonal containing the point cloud for contents in Categories 1 and 2. For Category 3, the intrinsic resolution is used, which in the `dmetric` is defined as the maximum distance between neighboring voxels in the original point cloud.

2.4.1.3 End-to-end color metrics

After finding a point-to-point correspondence using (2.6) and (2.5), texture (or color) metrics for point clouds are calculated the same way MSE is performed in images. Colors are first converted to YUV space following the BT.709 standard, as shown in (2.4), then the MSE is calculated separately for each channel,

$$\mathrm{MSE_Y} = \max\left\{\frac{1}{K_{\mathcal{O}}}\sum_{k=1}^{K_{\mathcal{O}}}\left(\mathrm{Y}_{\mathcal{O}}(k) - \mathrm{Y}_{\mathcal{D}}(k_{\mathcal{O}})\right)^2, \frac{1}{K_{\mathcal{D}}}\sum_{k=1}^{K_{\mathcal{D}}}\left(\mathrm{Y}_{\mathcal{D}}(k) - \mathrm{Y}_{\mathcal{O}}(k_{\mathcal{D}})\right)^2\right\}, \tag{2.15}$$

$$\mathrm{MSE_U} = \max\left\{\frac{1}{K_{\mathcal{O}}}\sum_{k=1}^{K_{\mathcal{O}}}\left(\mathrm{U}_{\mathcal{O}}(k) - \mathrm{U}_{\mathcal{D}}(k_{\mathcal{O}})\right)^2, \frac{1}{K_{\mathcal{D}}}\sum_{k=1}^{K_{\mathcal{D}}}\left(\mathrm{U}_{\mathcal{D}}(k) - \mathrm{U}_{\mathcal{O}}(k_{\mathcal{D}})\right)^2\right\}, \tag{2.16}$$

$$\mathrm{MSE_V} = \max\left\{\frac{1}{K_{\mathcal{O}}}\sum_{k=1}^{K_{\mathcal{O}}}\left(\mathrm{V}_{\mathcal{O}}(k) - \mathrm{V}_{\mathcal{D}}(k_{\mathcal{O}})\right)^2, \frac{1}{K_{\mathcal{D}}}\sum_{k=1}^{K_{\mathcal{D}}}\left(\mathrm{V}_{\mathcal{D}}(k) - \mathrm{V}_{\mathcal{O}}(k_{\mathcal{D}})\right)^2\right\}. \tag{2.17}$$

It is worth noting that neighboring points within the same distance have their colors averaged for the MSE computation.

Color PSNR is calculated by

$$\mathrm{PSNR_Y} = 10\log_{10}\left(\frac{p_c^2}{\mathrm{MSE_Y}}\right), \tag{2.18}$$

$$\mathrm{PSNR_U} = 10\log_{10}\left(\frac{p_c^2}{\mathrm{MSE_U}}\right), \tag{2.19}$$

$$\mathrm{PSNR_V} = 10\log_{10}\left(\frac{p_c^2}{\mathrm{MSE_V}}\right). \tag{2.20}$$

Where $p_c = 1$ when considering a floating-point representation, or $p_c = 255$ when considering an 8-bit color depth integer representation.

A common method for considering all the color components together was proposed by Ohm *et. al* for the comparison of video coding standards [86], and it has been used for point cloud assessment studies [6], [80]. It consists of using the weighted average of each PSNR channel, such that the luma and the chroma measurements can be gathered in a single value:

$$\mathrm{PSNR_{YUV}} = \frac{6\,\mathrm{PSNR_Y} + \mathrm{PSNR_U} + \mathrm{PSNR_V}}{8}. \tag{2.21}$$

2.4.2 Projection-based Metrics

As considered in the volume visualization pipeline of Figure 2.4, point clouds are in fact consumed as 2D projections, and therefore it makes sense to evaluate its quality using 2D projections. This approach was first used in [87], where the rendered point clouds were mapped onto planar surfaces, then 2D image quality assessment metrics were used to evaluate compression performance. Projection-based metrics are interesting because they provide a quality metric that considers the point cloud as a whole, differently from the point-based metrics that consider the geometry and the attributes separately. Also, for applications with a well-defined rendering methodology and well-known usages (i.e., which viewpoints are more important, the most used camera settings for visualization, etc.), projection-based metrics can be fine-tuned to mimic those specific conditions to better approximate the human subjective perception of the contents. In studies considering subjective quality assessment, for example, the same rendering technique is applied for subjective and projection-based metrics [6]. Likewise, in such studies, it is preferred to use a large number of viewpoints to capture as much visual information as possible [80], since the aliasing present in unaligned-axis projections will also be present in the subjective assessment.

For a more generic application, however, a rendering approach that requires the least processing of the voxels in the projections is preferred. This means using only the six axis-aligned orthographic projections. This way, it is possible to have a one-to-one correspondence between voxels and pixels. As a result, the camera parameters are set to acquire projections that are perfectly aligned with the bounding cube containing the point cloud. This ensures that a point cloud of depth 10, for example, will have six bitmap images of 1024-by-1024 pixels, as illustrated by Figure 2.20. As point clouds are inherently sparse, most of the volume of the bounding cube is empty. Hence, most of the pixels of the projected bitmaps do not belong to the effective part of the content (i.e., background color), so it is important to choose a value of background color to least interfere with the calculations. Moreover, it has been found that excluding part of the background pixels improves the accuracy of the metric prediction [88]. This way, image quality metrics are applied to the six pairs of projections, and the total score is computed as the average of the metrics. In this work, three different 2D metrics were selected: the MSE/PSNR, which is based on pixels differences and relates with the point-based metrics of the last Section; Structural Similarity Index Measure (SSIM), which searches for similarities in patches rather then pixel-wise; and the Visual Information Fidelity (VIF) measure, which is an information theory approach to model how image information is perceived in the HVS [89].

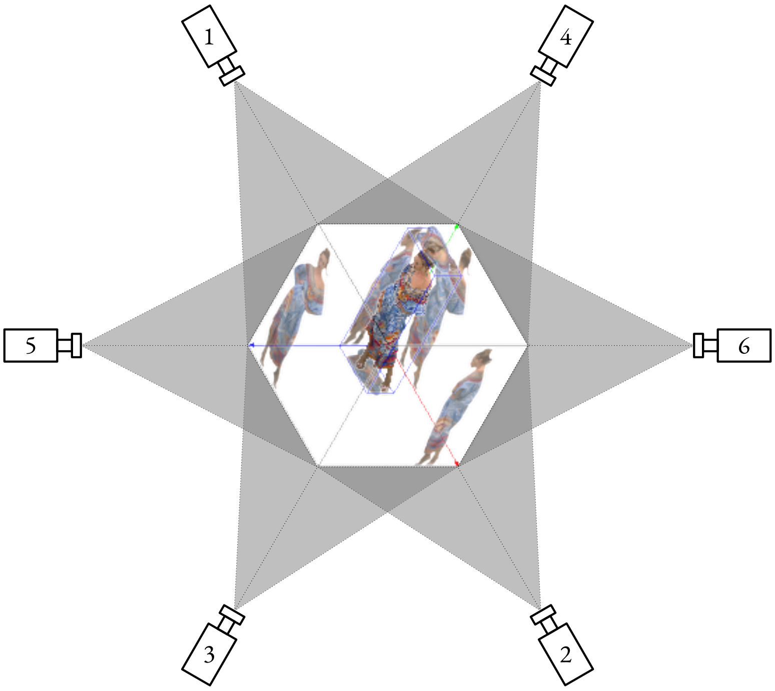


Figure 2.20: The six camera positions used to get axis-aligned projections.

2.4.2.1 Projected MSE and PSNR

MSE is used for measuring the differences of pixels between a reference image and its distorted version. More specifically, its scale-agnostic version, the PSNR, is widely used for evaluating the degradation of compression artifacts of images. The MSE computation of an RGB image is analogous to the one presented in Section 2.4.1. The error vector's squared norm in the RGB space between each pixel in the reference image and its counterpart in the distorted image is computed. Thus, to calculate the projected MSE (PMSE) of a point cloud, we average the MSE of the six pairs of W_{n_v} by H_{n_v} images:

$$\text{PMSE} = \frac{1}{6}\sum_{n_v=1}^{6}\left(\frac{1}{W_{n_v}H_{n_v}}\sum_{x=1}^{W_{n_v}}\sum_{y=1}^{H_{n_v}}||\text{RGB}_{\text{ref}}^{n_v}(x,y) - \text{RGB}_{\text{dist}}^{n_v}(x,y)||^2\right). \tag{2.22}$$

The projected PSNR (PPSNR), for 8-bit color depth images, is defined as,

$$\text{PPSNR} = 10\log_{10}\left(\frac{255^2}{\text{PMSE}}\right). \tag{2.23}$$

2.4.2.2 Projected SSIM

The SSIM assesses perceptual quality between reference and distorted images by comparing local similarities on spatial patches, assuming that the HVS is highly adapted to extract structural information from visual scenes. It was created from the perspective of image formation [90], so it performs three comparisons (luminance, contrast, and structure) on each patch, and then averages the comparisons over the whole image to arrive at an overall image quality, which is referred to as the mean SSIM (MSSIM). As the SSIM was developed for grayscale images, we use it on the luma channel Y only. Following the implementation of SSIM [91], in its default configuration, we have that the index between signals x and y, corresponding to 11-by-11 Gaussian windows with $\sigma = 1.5$ in each image is:

$$\text{SSIM}(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \tag{2.24}$$

where μ represents the mean, σ^2 the variance, $C_1 = (0.01 \cdot 255)^2$, and $C_2 = (0.03 \cdot 255)^2$. The MSSIM is, then, the average of the M patches over the entire images X and Y,

$$\text{MSSIM}(X,Y) = \frac{1}{M}\sum_{j=1}^{M}\text{SSIM}(x_j, y_j). \tag{2.25}$$

Finally, the projected SSIM (PSSIM) is calculated as the average of the MSSIM for every viewpoint projection pair,

$$\text{PSSIM} = \frac{1}{6}\sum_{n_v=1}^{6}\text{MSSIM}(X_{n_v}, Y_{n_v}). \tag{2.26}$$

2.4.2.3 Projected VIF

The information theory approach of VIF [92] measures the fidelity of a distorted image by calculating the loss of image information to the distortion process and comparing it with its reference counterpart. Thus, exploring the relationship between image information and visual quality. Figure 2.21 depicts a block diagram of the information content model used in VIF.

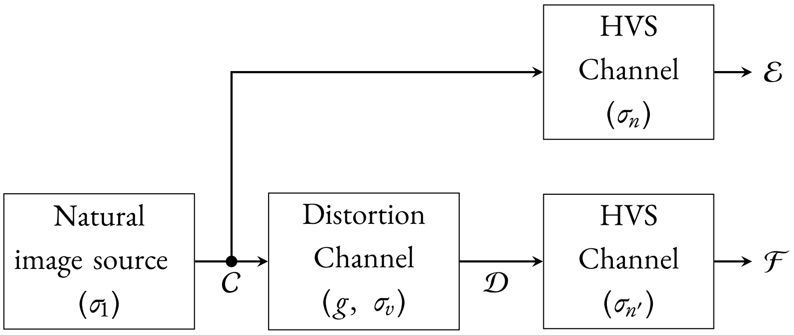

HVS Channel (σ_n)
$\mathcal{E}$
Natural image source (σ_1)
C
Distortion Channel (g, σ_v)
$\mathcal{D}$
HVS Channel ($\sigma_{n'}$)
$\mathcal{F}$

Figure 2.21: Information content model used in VIF [89].

The reference image is the output of a stochastic "natural" source, thus, modeled as a random field C using Gaussian Scale Mixtures (GSM), which, after passing through the HVS channel, becomes $\mathcal{E}$, the information processed by the brain. The information content of the reference image is quantified as being the mutual information between C and $\mathcal{E}$, $I(C;\mathcal{E})$. Ideally, this would represent the information that the brain could extract from the output of the HVS. The information content of the distorted image is likewise quantified, but a distortion channel $\mathcal{D}$ is introduced before the HVS, such that it becomes the mutual information between C and $\mathcal{F}$, $I(C;\mathcal{F})$. The VIF measure is, then, calculated as the ratio of the sum of the distorted image information content over the sum of the reference image reference content, across all subbands of the GSM [89], [93],

$$\text{VIF} = \frac{\sum_{j\in\text{ subbands}} I(C^j;\mathcal{F}^j|s^j)}{\sum_{j\in\text{ subbands}} I(C^j;\mathcal{E}^j|s^j)}. \tag{2.27}$$

In this work, we use a computationally simpler multi-scale pixel implementation of VIF, the pixel domain VIF (VIFP) [91]. Similar to the SSIM, the VIF was designed for grayscale images only, so again it is applied only for the luma channel. The mutual information is calculated across four subbands. For each subband j, M_j Gaussian windows are considered, such that

$$\text{VIFP} = \frac{\sum_{j=1}^{4}\sum_{i=1}^{M_j}\log_{10}\frac{1+g(i,j)^2\sigma_1(i,j)^2}{\sigma_v(i,j)^2+\sigma_n(i,j)^2}}{\sum_{j=1}^{4}\sum_{i=}^{M_j}\log_{10}\frac{1+\sigma_1(i,j)^2}{\sigma_n(i,j)^2}}. \tag{2.28}$$

The projected VIFP (PVIFP) is, then, the average of the VIFP calculated for projection pair,

$$\text{PVIFP} = \frac{1}{6}\sum_{n_v=1}^{6}\text{VIFP}_{n_v}. \tag{2.29}$$

2.5 POINT CLOUD PROCESSING

Point cloud processing is used in several applications. One way to achieve lossy geometry compression, for example, is by downsampling the input point cloud, as done by TMC13 (Figure 2.14). Some rendering applications require to downsample a point cloud when computer resources are limited. Alternatively, when computer resources are not a problem, and high-perceived quality is required, upsampling or super-resolving a point cloud could be of interest. When trying to improve the perceived quality of point clouds, some techniques to smooth its geometry and/or attributes, and to fill in holes may prove useful.

In this Section, the processing techniques utilized in this work are briefly explained.

2.5.1 Downsampling

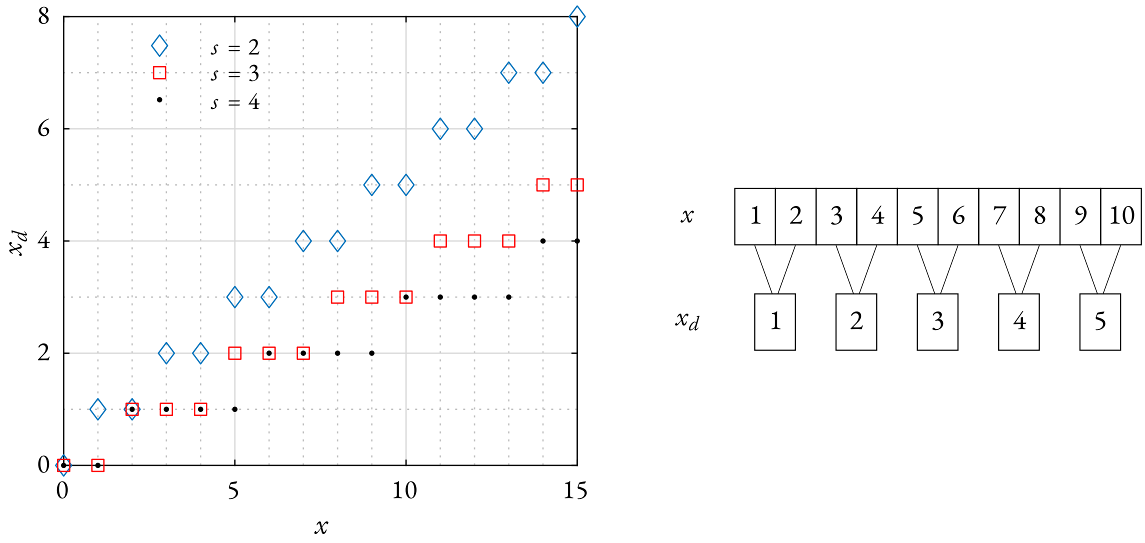
The downsampling of point clouds can be approached in two ways: grid or set downsampling. In grid downsampling, the decimation is done in the voxel grid, the bounding cube. The point cloud is downsampled as a consequence of this process. In set downsampling, the decimation is performed directly in the geometry set of points, not changing the voxel grid in the process.

2.5.1.1 Grid Downsampling

Grid downsampling makes sense for point clouds bound to a voxel-grid, i.e., voxelized point clouds. To perform the downsampling, the point cloud geometry is firstly scaled and then re-voxelized. This can be seen as a new subdivision of the cubic volume containing the point cloud but with larger voxels. In the scaling step, all the original bounding cube's voxels are grouped into larger s-sided voxels. Equivalently, the point cloud geometry can be scaled by $^1/_s$, maintaining the unitary volume of the voxel. The voxelization step is achieved by rounding each scaled coordinate to the nearest integer value. Duplicate points, which may arise after the rounding step, are dealt with the same way as in the voxelization process of Section 2.3.2. They are consolidated, and their attributes averaged.

Using some auxiliary functions [94]:

$$\begin{aligned}
\texttt{sign}(x) &= \begin{cases} 1, x \geq 0 \\ -1, x < 0 \end{cases}, && (2.30)\\
\texttt{floor}(x) &= \lfloor x \rfloor = \text{the greatest integer less then or equal to } x, && (2.31)\\
\texttt{ceil}(x) &= \lceil x \rceil = \text{the least integer greater then or equal to } x, && (2.32)\\
\texttt{round}(x) &= \texttt{sign}(x)\lfloor |x| + 0.5 \rfloor = \left\lceil \frac{\lfloor 2x \rfloor}{2} \right\rceil, && (2.33)\\
\texttt{unique}(X) &= \text{the set containing only the unique vectors from } X. && (2.34)
\end{aligned}$$



(a)

(b)

Figure 2.22: One-dimensional analysis of the grid downsampling for integer values of s.

The grid-downsampled geometry is defined as,

$$V_d = \texttt{unique}\left(\texttt{round}\left(\frac{1}{s}V\right)\right), \tag{2.35}$$

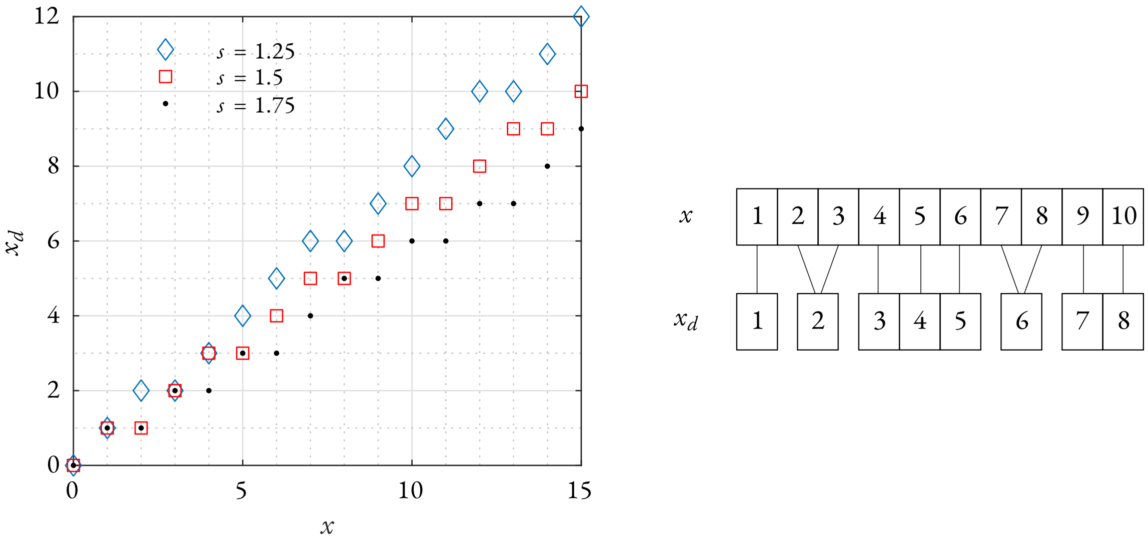
where the scalar multiplication and the $\texttt{round}(\cdot)$ function are applied on each vector component.

Analyzing the scaling function in one dimension, $x_d = \texttt{round}(x/s)$, for integer values of s in Figure 2.22(a), we notice that, except for boundary points, every s values in x are reduced to one value in x_d. Or, equivalently, every value in x_d maps s values in x, as depicted in Figure 2.22(b), where the downsampling is represented as a hierarchical tree with x_d representing parent nodes, and x representing child nodes for $s = 2$. Moreover, for integer values of s, the number of child nodes is constant for all parent values. In 3D, this means that each parent voxel from the downsampled bounding cube maps s^3 children in the original bounding cube. This represents a pruning of the original octree structure, in fact when $s = 2^n$, $n = 1, 2, 3, \ldots$, the pruning occurs exactly[2] at level $d - n$. Figure 2.24 shows a point cloud rendered with different octree prunings.

For non-integer values of s, the number of children per parent node is not regular, it varies depending on x_d, as depicted in Figure 2.23(a). In 3D, this means that each parent voxel from the downsampled bounding cube maps *up to* s^3 children in the original bounding cube. Specifically, when $\{s \in \mathbb{Q} \mid 1 < s < 2\}$, the downsampling is only partial. There are parent nodes with only one child (uniparous), and parent node with two children (multiparous), as shown in Figure 2.23(b). In this case, parent nodes may map 1, 2, 4, or 8 children depending on the combination of uniparous and multiparous parents on each (x_d, y_d, z_d) coordinates.

Briefly, the grid downsampling is a fast method representing a pruning of the octree structure at

[2] The term *exactly* is somewhat relaxed here since the $\texttt{round}$ function adds an approximation that would not occur when n levels are removed from the octree. To get the strictly exact pruning equivalence, Equation (2.35) should be defined using the $\texttt{floor}$ function instead.



(a)

(b)

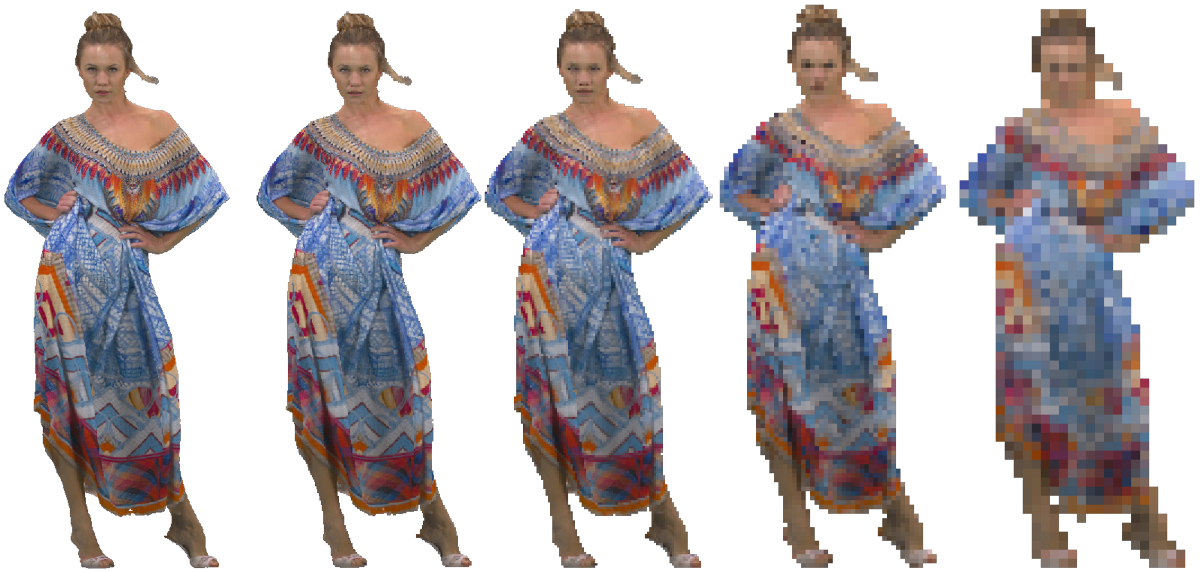
Figure 2.23: One-dimensional analysis of the grid downsampling for fractional values of s.

an exact or partial level. This octree pruning ensures that spatial correlation among occupied voxels remains the same or even increases (due to the grouping of voxels), which is a desired property for compression purposes. However, the quantization step of this method not only removes duplicate points, but also causes a slight shift in the geometry. When comparing the expanded geometry, $s \cdot V_d$, with V, for example, we will find $\text{D1}_{\mathcal{D}}^{\text{MSE}} > 0$, meaning that even the remaining points in V_d cannot be directly upsampled without error (shown in Figure 2.24). This fact makes super-resolving grid downsampled point clouds more challenging.

2.5.1.2 Set Downsampling

In set downsampling, points are usually decimated using some distance-based criterion or even random decimation. The advantage of these downsampling methods is that the points which remain do not need to be re-voxelized, meaning that they are correct points from the original point cloud, $\text{D1}_{\mathcal{D}}^{\text{MSE}} = 0$, which is a desirable property for interpolation. However, these methods tend to decrease the points' spatial correlation, making the point cloud compression harder. This happens because loads of points become isolated, and those points need more bits to be encoded compared to grouped points.

Two examples of set downsampling used for point clouds are the LoD representation utilized in TMC13 (Figure 2.18), and the Poisson disk sampling [95], where samples are at least distance r apart from some user-supplied density parameter r. Both cases require that distances from each point to its neighbors to be calculated, which adds complexity and makes them slower than grid downsampling. Figure 2.25 depicts the Poisson disk sampling applied for different values of r, which were chosen in order to generate a similar number of points of the grid downsampling in Figure 2.24. The voxels' size was increased for illustration purposes, and the D1 metric was calculated for comparison between the methods. We also calculated the rate in bits per occupied voxel (bpov) needed to create an octree representation for each case, to illustrate the diminishing of spatial correlation.



857,966 points
$\mathrm{D1}_{\mathcal{O}}^{\mathrm{MSE}} = 0$
$\mathrm{D1}_{\mathcal{D}}^{\mathrm{MSE}} = 0$
2.99 bpov

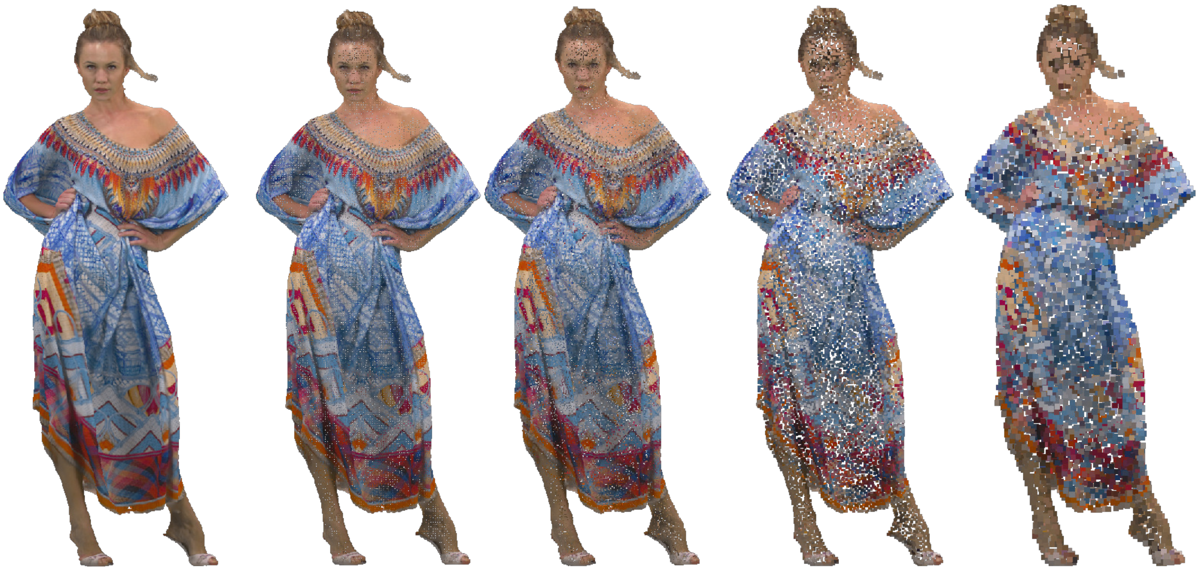
238,492 points
$\mathrm{D1}_{\mathcal{O}}^{\mathrm{MSE}} = 1.50$
$\mathrm{D1}_{\mathcal{D}}^{\mathrm{MSE}} = 0.68$
2.78 bpov

62,130 points
$\mathrm{D1}_{\mathcal{O}}^{\mathrm{MSE}} = 4.50$
$\mathrm{D1}_{\mathcal{D}}^{\mathrm{MSE}} = 2.25$
2.67 bpov

15,688 points
$\mathrm{D1}_{\mathcal{O}}^{\mathrm{MSE}} = 16.56$
$\mathrm{D1}_{\mathcal{D}}^{\mathrm{MSE}} = 9.31$
2.59 bpov

3,781 points
$\mathrm{D1}_{\mathcal{D}}^{\mathrm{MSE}} = 64.65$
$\mathrm{D1}_{\mathcal{D}}^{\mathrm{MSE}} = 38.46$
2.49 bpov

Figure 2.24: Downsampling at exact octree levels. The original depth-10 point cloud followed by octree pruning with depth $d = 9, 8, 7,$ and 6, which corresponds to a downsampling using $s = 2^1, 2^2, 2^3,$ and 2^4, respectively.



857,966 points
$\mathrm{D1}_{\mathcal{O}}^{\mathrm{MSE}} = 0$
$\mathrm{D1}_{\mathcal{D}}^{\mathrm{MSE}} = 0$
2.99 bpov

345,281 points
$\mathrm{D1}_{\mathcal{O}}^{\mathrm{MSE}} = 0.60$
$\mathrm{D1}_{\mathcal{D}}^{\mathrm{MSE}} = 0$
6.77 bpov

64,176 points
$\mathrm{D1}_{\mathcal{O}}^{\mathrm{MSE}} = 2.21$
$\mathrm{D1}_{\mathcal{D}}^{\mathrm{MSE}} = 0$
16.01 bpov

15,972 points
$\mathrm{D1}_{\mathcal{O}}^{\mathrm{MSE}} = 9.13$
$\mathrm{D1}_{\mathcal{D}}^{\mathrm{MSE}} = 0$
23.58 bpov

5,023 points
$\mathrm{D1}_{\mathcal{O}}^{\mathrm{MSE}} = 26.28$
$\mathrm{D1}_{\mathcal{D}}^{\mathrm{MSE}} = 0$
30.24 bpov

Figure 2.25: Downsampling using Poisson disk sampling. The minimum distance between samples at each model is $r = \sqrt{2},\ 3,\ 3\sqrt{3},$ and $7\sqrt{2}$, respectively.

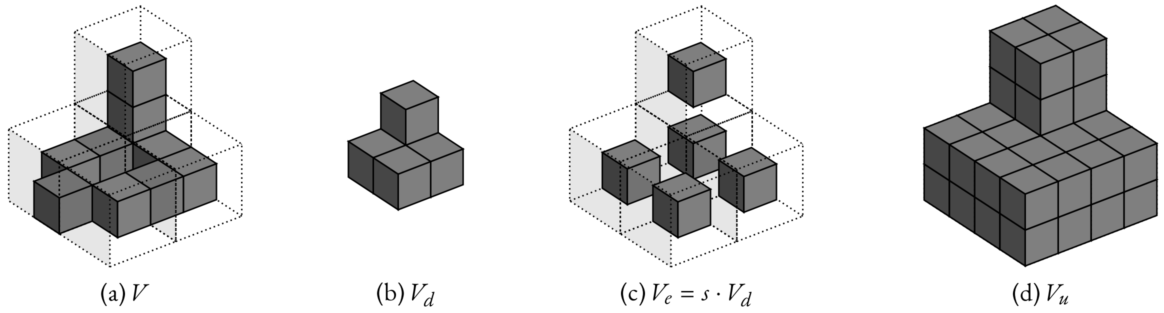

(a) V (b) V_d (c) $V_e = s \cdot V_d$ (d) V_u

Figure 2.26: Down- and upsampling representations for $s = 2$. (a) Child nodes surrounded by its parent-nodes. (b) Parent nodes. (c) Expansion of V_d. (d) NNI upsampling.

2.5.2 Upsampling

Point cloud upsampling is the inverse operation of the downsampling process of Section 2.5.1. It seeks to increase the number of voxels by trying to re-create points decimated during the downsampling process. In the grid downsampling case, this means first to expand the downsampled geometry, i.e., $s \cdot V_d$, refine the expanded geometry and finally interpolate the missing points. For the set downsampling, neither the expansion nor the refinement steps are necessary, and just the interpolation step is needed.

Upsampling 3D points is a much harder problem than the downsampling one, and when considering specifically voxelized point clouds with texture information, there are few works about it. When the upsampling technique achieves good results, it is usually referred to as super-resolution (SR). The classic approach is to obtain a global function (or set of local functions) that approximates V_d in the least-squares sense, then solve the minimization problem in order to interpolate values between points [96]. The moving least squares (MLS) method was used in a few works considering point sets [97]–[99], but no texture nor voxelized point clouds were considered. Another approach is to use graph signal processing (GSP) to perform the interpolation. Dinesh *et. al* used graph total variation (GTV) on surface normals to obtain interesting results for super-resolving the geometry of point clouds [100]. The method was later improved [101], and voxelized point clouds with texture were considered, however, the SR method was only applied to LR point clouds created using Poisson disk sampling. Attempting a different strategy, Hamid-Cherif *et. al* [102] used local similarities and estimate surface normals to super-resolve point sets, without considering voxelized point clouds with texture information, however.

When voxelized point clouds are considered, the problem becomes more constrained since there is a finite number of possible values for the missing points. When considering grid downsampling LR point clouds, the simplest way to perform the upsampling is by using the nearest neighbor interpolation (NNI). Due to the way the downsampling is performed, the location of all possible children from each parent node can be inferred, thus, NNI is performed by setting all these children as occupied. Figure 2.26 illustrates what happens to the original geometry as it is downsampled and then upsampled using NNI.

Other image processing resampling methods (e.g., linear, or cubic interpolation) could also be adapted for 3D points. However, the 3D problem for geometry upsampling deals with binary data. Voxels are either empty or occupied, so methods that propose some kind of average cannot be directly used with binary data. Garcia *et. al* [5] super-resolved grid downsampled LR point clouds containing texture, using examples from full-resolution prior frames [103]. Later, three inter-frame geometry SR methods were developed to provide context for coding point clouds sequences. In Chapter 3, we expand one of these methods based on self-similarities of LR point clouds to work for the intra-frame case and generalize it for non-exact octree downsamplings.

2.5.3 Smoothing

Smoothing techniques are useful to reduce high-frequency noise of point clouds. The Laplacian smoothing (LS) [104] is an algorithm used to smooth meshes that can easily be adapted to work on point clouds. The idea is to update the position of each occupied voxel by averaging the positions of occupied neighbor voxels in a 3 × 3 × 3 neighborhood. Defining $\mathcal{N}_3(k)$ as the subset containing all occupied voxels in a 3 × 3 × 3 neighborhood around voxel $\mathbf{v}(k)$, the updated coordinate is

$$\mathbf{v}_{LS}(k) = \texttt{round}\left(\frac{1}{K_{\mathcal{N}}+1}\left(\mathbf{v}(k) + \sum_{\substack{\forall i:\\ \mathbf{v}(i)\in\mathcal{N}_3(k)}} \mathbf{v}(i)\right)\right), \tag{2.36}$$

where $K_{\mathcal{N}}$ is the number of occupied neighbors in the searched neighborhood. The color associated with each voxel is interpolated using the inverse-distance, ∂^{-1}, between the current voxel $\mathbf{v}_{LS}(k)$ and each of its neighbors in $\mathcal{N}_3(k)$,

$$\mathbf{c}_{LS}(k) = \begin{cases} \mathbf{c}(k) & \text{, if } \mathbf{v}(k) = \mathbf{v}_{LS}(k), \\ \dfrac{\sum\limits_{\substack{\forall i:\\ \mathbf{v}(i)\in\mathcal{N}_3(k)}} \partial_i^{-1}\mathbf{c}(i)}{\sum\limits_{\substack{\forall i:\\ \mathbf{v}(i)\in\mathcal{N}_3(k)}} \partial_i^{-1}} & \text{, otherwise.} \end{cases} \tag{2.37}$$

After smoothing every voxel, duplicate points that eventually appear have their colors averaged,

$$C_{LS} = \bigcup_{k=1}^{K} \texttt{round}\left(\frac{1}{D+1}\left(\mathbf{c}_{LS}(k) + \sum_{\substack{\forall i:\\ \mathbf{v}(i)=\mathbf{v}(k)}} \mathbf{c}_{LS}(i)\right)\right), \tag{2.38}$$

where D indicates the number of duplicates of voxel $\mathbf{v}(k)$. Then, duplicate points are consolidated,

$$V_{LS} = \texttt{unique}\left(\bigcup_{k=1}^{K} \mathbf{v}_{LS}(k)\right). \tag{2.39}$$

An illustration of the LS is depicted in Figure 2.27.

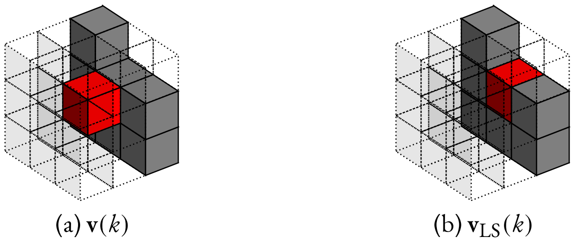

(a) $\mathbf{v}(k)$ (b) $\mathbf{v}_{\text{LS}}(k)$

Figure 2.27: Laplacian smoothing illustration. The voxel in the center of the $3 \times 3 \times 3$ neighborhood in (a) gets its position updated after the LS is applied in (b).

A more general approach consists of taking any 3D kernel, instead of the $3 \times 3 \times 3$ cube used in the LS, and applying different weights to each neighboring voxel. The filtered voxel position would then be the weighted average of the occupied voxels inside the kernel. This way, it is possible to filter both geometry and color in the same fashion 2D kernel filtering is done in images. Notice, however, that differently from 2D filtering, where the kernel is constant, the effective number of voxels in the 3D kernel will vary depending on the neighborhood's voxel occupancy.

2.5.4 Morphological Transformations

The last processing techniques covered in this work are the morphological transformations. 3D morphological transformations are operations that change the geometry structure. Besides the input geometry, the transformations require a structuring element, a 3D structure representing a fixed voxel neighborhood. The basic morphological operators are erosion and dilation, then combinations of the two operators can be used to achieve different transformations.

The erosion operator compares each voxel's neighborhood with the structuring element. Only the voxels whose neighborhood fully intersects with the given 3D structure are kept. All the others are removed (eroded).

The dilation operator does the opposite. It takes the union of each voxel's neighborhood with the structuring element. Thus, increasing the number of voxels, dilating the geometry. The `unique(·)` function is again used to consolidate duplicate points. The newly created points' color is interpolated using the weighted average of its neighboring voxels' colors. Figure 2.28 illustrates the basic morphological operations using a $3 \times 3 \times 3$ block as the structuring element.

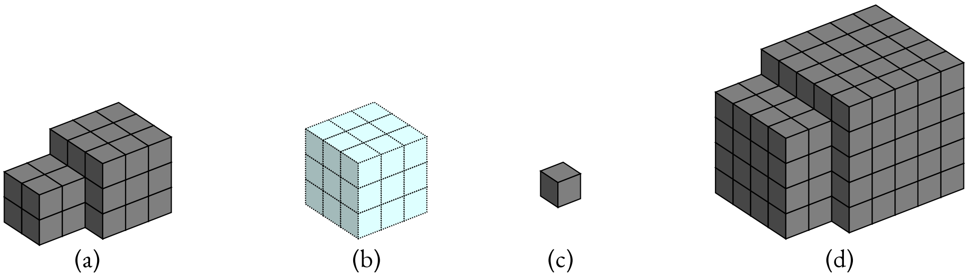

(a) (b) (c) (d)

Figure 2.28: Basic morphological operations. (a) Input point cloud. (b) Structuring element. (c) Eroded geometry. (d) Dilated geometry.

The closing operation is performed by first dilating the geometry, then eroding it using the same structuring element. It is useful for filling in holes in the geometry of a point cloud. When filling in holes caused by an upsampling method, this operation can be further improved by removing added points that do not belong to the NNI geometry, i.e., points we are certain that do not exist in the original point cloud.

3 METHODOLOGY

"Omnis cellula e cellula."

—Rudolf Virchow, *Cellular Pathology,* 1860

3.1 THE RELEVANCE OF FRACTIONAL RESAMPLING

In this Chapter, we explore the different strategies developed during this work to super-resolve point clouds. The premise was to develop an SR method for voxelized point clouds. Moreover, since the state-of-the-art codec employed for this kind of point cloud (TMC13) utilizes the octree to represent the geometry, we decided to focus only on LR versions that are efficiently represented by this data structure. For this reason, we only considered grid downsampling since it maintains, or even increases, the point's spatial correlation in the LR point clouds. Table 3.1 summarizes the upsampling techniques presented in Chapter 2. We can see that grid resampling was not much explored. In fact, only Garcia *et al.* did it. Still, only the exact octree pruning resampling was explored, and only for the inter-frame case.

Table 3.1: Summary of point cloud resampling strategies found in the literature.

Downsampling		Proposed upsampling	Voxelized Point Clouds	Texture
Grid	$s \in \mathbb{N} \mid s = 2^n$	Garcia *et al.* [5], [103]	✓	✓
	$s \in \mathbb{Q}$	–	–	–
Set		MLS: [97]–[99]	✗	✗
		GTV: [100], [101]	✓†	✓†
		Local similarities: [102]	✗	✗

† Partially.

Fractional resampling of voxelized point clouds, although utilized in TMC13, has never been theoretically explored. Using fractional scale factors in point cloud downsampling allows for more flexible octree pruning. By understanding the properties and irregularities of this kind of downsampling, we were able to propose the upsampling methods presented in this Chapter.

The basic idea explored for super-resolving voxelized point clouds is that there is a finite number of possible children. Child voxels can only come from occupied parent voxels. Therefore, all the presented methods share the same goal: find a way to choose which child nodes to keep (or to remove) given the NNI upsampled geometry of an input point cloud. To this extent, first, we define some concepts of the NNI upsampling, then those concepts are employed in the SR methods. Finally, we present a web-based renderer, developed for a future subjective assessment test of the proposed method.

3.2 NEAREST NEIGHBOR INTERPOLATION UPSAMPLING

In Section 2.5.2, we saw that the NNI upsampling is achieved by setting to occupied all possible children of each parent node. Here, we formalize how this operation is performed. All the operations presented hereon assume that voxels can only have integer-valued components.

Let $\mathcal{V}_u(k) = \{\mathbf{v}_u(i)\}$ be the set containing all possible child nodes from $\mathbf{v}_d(k)$, such that,

$$\texttt{round}(\mathbf{v}_u(i)/s) = \mathbf{v}_d(k), \tag{3.1}$$

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

$$\mathbf{v}_u(i) = \texttt{round}(s \cdot \mathbf{v}_d(k)) + \epsilon(i), \tag{3.2}$$

where $\epsilon(i)$, $\{\epsilon(i) \in \mathbb{Z} \mid |\epsilon(i)| \leq \lceil s \rceil - 1\}$, is needed to account for the information lost due to the round function, and it can be found by reproducing the downsampling and checking the parent-child mapping, as in Figures 2.22 and 2.23. Let also $\mathcal{E}(k) = \{\epsilon(i)\}$ be the set containing all possible rounding errors for parent node $\mathbf{v}_d(k)$, then we can rewrite (3.2) in matrix form as

$$\mathcal{V}_u(k) = \texttt{round}(s \cdot \mathbf{v}_d(k)) + \mathcal{E}(k). \tag{3.3}$$

For example, considering $s = 7/4 = 1.75$,

$$\begin{array}{rcl|cc|cc|cc|c|cc|cc|cc|c|l}
x & = & [0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15], \\
x_d & = & [0 & 1 & 1 & 2 & 2 & 3 & 3 & 4 & 5 & 5 & 6 & 6 & 7 & 7 & 8 & 9], \\
\texttt{round}(s \cdot x_d) & = & [0 & 2 & 2 & 4 & 4 & 5 & 5 & 7 & 9 & 9 & 11 & 11 & 12 & 12 & 14 & 16], \\
\epsilon_x = x - \texttt{round}(s \cdot x_d) & = & [0 & \text{-}1 & 0 & \text{-}1 & 0 & 0 & 1 & 0 & \text{-}1 & 0 & \text{-}1 & 0 & 0 & 1 & 0 & \text{-}1].
\end{array}$$

Now, if we take the parent node $\mathbf{v}_d = (3, 4, 6)$, we have

$$\mathcal{V}_u\Big|_{\mathbf{v}_d=(3,4,6)} = (5,\ 7,\ 11) + \left\{\begin{matrix} (0, & 0, & -1) \\ (0, & 0, & 0) \\ (1, & 0, & -1) \\ (1, & 0, & 0) \end{matrix}\right\} = \left\{\begin{matrix} (5, & 7, & 10) \\ (5, & 7, & 11) \\ (6, & 7, & 10) \\ (6, & 7, & 11) \end{matrix}\right\}. \tag{3.4}$$

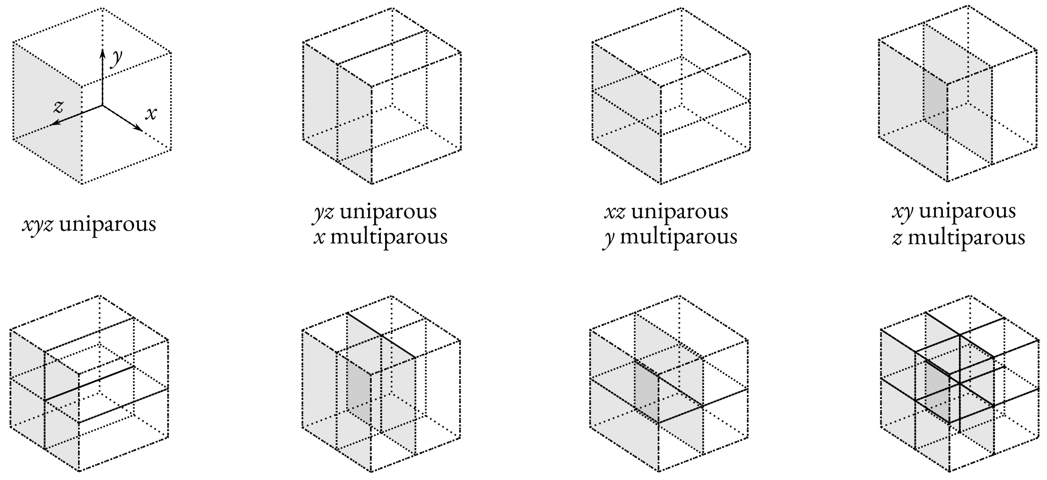
Therefore, the NNI upsampled geometry is defined as

$$V_u = \bigcup_{k=1}^{K} \mathcal{V}_u(k). \tag{3.5}$$

The colors from the children in $\mathcal{V}_u(k)$ are defined in the subset $C_u(k) = \{\mathbf{c}_u(i)\}$, such that,

$$\mathbf{c}_u(i) = \mathbf{c}_d(k),\ \forall i, \text{ and,} \tag{3.6}$$

$$C_u = \bigcup_{k=1}^{K} C_u(k). \tag{3.7}$$



xyz uniparous

yz uniparous
x multiparous

xz uniparous
y multiparous

xy uniparous
z multiparous

z uniparous
xy multiparous

y uniparous
xz multiparous

x uniparous
yz multiparous

xyz multiparous

Figure 3.1: The eight types of parent-children conditions found for $1 < s < 2$. Each parent voxel is classified as such before performing the NNI upsampling.

As we saw on Section 2.5.1.1, the number of children, $i_{\max}$, depends on the scale factor s, the number of uniparous[1] components a, and the number of multiparous components b, such that,

$$i_{\max} = (\lceil s \rceil - 1)^a \lceil s \rceil^b. \tag{3.8}$$

So it is useful to label parent voxels according to the capacity to map descendants from their position's components. We can divide V_d into eight subsets, $\{V_{d,j}\}$, depending on the label of each parent voxel, as follows:

0: all components are uniparous,
$(\lceil s \rceil - 1)^3$ children;

1: only x component is multiparous,
$(\lceil s \rceil - 1)^2 \lceil s \rceil$ children;

2: only y component is multiparous,
$(\lceil s \rceil - 1)^2 \lceil s \rceil$ children;

3: only z component is multiparous,
$(\lceil s \rceil - 1)^2 \lceil s \rceil$ children;

4: only x component is uniparous,
$(\lceil s \rceil - 1) \lceil s \rceil^2$ children;

5: only y component is uniparous,
$(\lceil s \rceil - 1) \lceil s \rceil^2$ children;

6: only z component is uniparous,
$(\lceil s \rceil - 1) \lceil s \rceil^2$ children;

7: all components ar multiparous,
$\lceil s \rceil^3$ children.

This allows us to treat each case separately, which facilitates other upsampling methods. Figure 3.1, shows the eight possible types of parenthood for $\{s \in \mathbb{Q} \mid 1 < s < 2\}$.

[1] Here we extend the meaning of uniparous to indicate parents with fewer children then the multiparous ones.

3.3 TOWARDS SUPER-RESOLUTION OF VOXELIZED POINT CLOUDS

In this Section, we present three attempts to generate a better upsampling geometry from the NNI results. In the first method, we improve the NNI upsampling by smoothing its geometry, reducing the aliasing effect. The second method consists of removing points from the NNI upsampling using the normal vectors calculated in V_d. The last one is a score-based approach that utilizes the parent neighborhood to decide which children from the NNI upsampling to be kept.

3.3.1 Smoothed Nearest Neighbor Interpolation Upsampling

As discussed in Section 2.5.3, point cloud smoothing techniques can be used to reduce the aliasing effects from the NNI upsampling. This approach is straightforward; one needs to apply low-pass filtering in the NNI upsampled point cloud. However, filtering operations require adjusting the filter parameters to the data. This tweaking of the filters is usually done manually, and it has proved to be hard to find constant parameters that would minimize the output error for different input point clouds.

For the geometry, several 3D-kernels were tested, and the best results were found using a 3×3×3 block kernel with equal weights for all voxels, which is equivalent to the geometry from the LS, as in (2.36). Regarding the color, slightly better gains were observed when comparing C_{LS} with the color obtained by filtering C_u using only the six neighbors sharing a face with the current voxel, 3D a cross-shaped kernel weighted by the inverse distance between neighbor voxels and the center voxel. However, using C_{LS} is a good option due to its lower complexity.

3.3.2 Carving the Nearest Neighbor Interpolation Upsampling Using Normal Vectors

This method aims to remove child nodes from the NNI upsampling that grow in the approximate direction of the parent node's normal vector. This way, we try to flatten the local surface perpendicular to the parent node's normal direction.

First, we need to calculate the normal vector of each point in V_d, which is done using Hoppe's algorithm considering the 12 nearest neighbors [65]. Then, the normal vectors are quantized in 26 directions. Those are the directions linking the center of the current voxel with the center of each of its 26 neighbors in a $3 \times 3 \times 3$ neighborhood. We define a child node's growth vector as the unit vector between the parent and the child node's centers. Here, we visualize child nodes as the subdivision of the parent node into eight sub-nodes, such that parent and child nodes are in the same scale (like in Figure 3.3(a)). Finally, we take the dot product of each child node's growth vector with the quantized normal vector. The child node with the maximum dot product is removed. If the same value is observed for different children, then all of those are removed. Figure 3.2 illustrates the method.

This approach is problematic because normal vectors calculated from V_d are biased. They can-

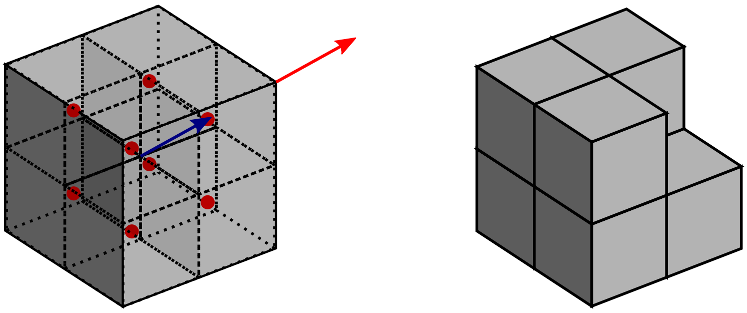


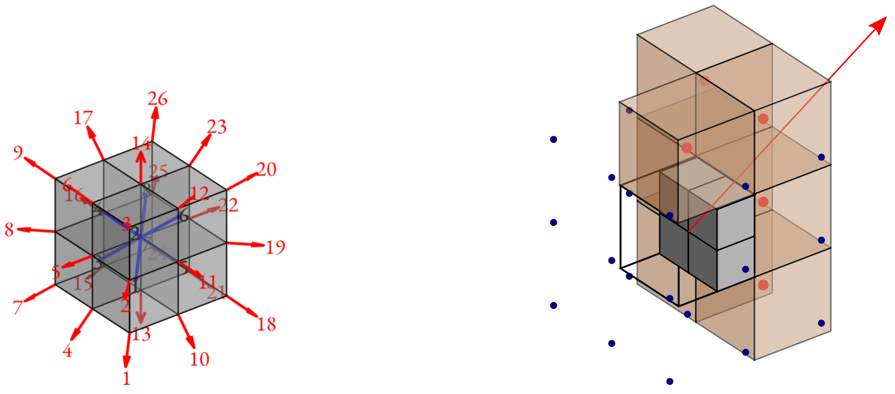
Figure 3.2: Normal carving illustration. In (a), we can see one of the child node's growth vector in blue and the quantized normal vector from the parent node in red. In (b), the resulting carved geometry is presented.

not give accurate information about the details of the point cloud. This can be illustrated by looking at a rock on a mountain slope. When we are close to the rock, we can differentiate the normal directions for the points representing its shape. When we are away from the mountain, the perceived normal from the rock becomes a single direction representing the mountain slope instead of the rock shape [84]. Thus we cannot use downsampled normals to assess information about the details of the original geometry. Another problem is that this method assumes that at least one child node must always be removed, which is not always true and may cause holes in the carved geometry.

3.3.3 Score-based Upsampling

This method uses the information from the parent node's neighbors (uncle nodes) to create a score for each child node from the NNI upsampling. Then, child nodes that have a score under a certain threshold are removed. A similar idea is used in TMC13's intra-prediction [62]. However, the score calculation is different.

The heuristic is that child nodes should only appear next to occupied uncle nodes. Nevertheless, choosing each uncle node's contribution amount to the final child score and the threshold value for which child nodes should be occupied is hard. The first approach we tried was to model the parent voxel as a nucleus that could be divided into smaller particles depending on the forces exerted by neighboring voxels. Occupied neighboring voxels exert a pulling force, while empty ones exert a repelling force, and the force magnitude is proportional to the inverse distance. Considering a $3 \times 3 \times 3$ neighborhood, there are 26 possible force directions (Figure 3.3(a)). Thus, first, we take the resultant force acting on the nucleus, then we project this force on each of the growth vectors indicating possible child nodes. If a projection has a zero or a positive value in a given growth direction, it indicates that a particle from the nucleus is attracted to that direction. Thus, a child node should exist there. Figure 3.3(b) illustrates the selected children using this score-based SR method. In practice, however, we found that the threshold value should be adaptive. By providing the information about the number of children each parent node should have (using information from the original point cloud), we found out that this method could provide educated guesses



(a)

(b)

Figure 3.3: Illustration of the score-based method modeling the parent voxel as a divisible nucleus. In (a), the 26 directions, and the 8 growth directions are depicted. In (b), the resulting force in red and the chosen children are shown.

about the children's occupancy. Still, we could not find a way to adaptively update each parent's threshold value without using extra information.

The second approach was simpler than the previous. Only neighbors sharing a face with the parent node in a 3 × 3 × 3 neighborhood were used. In this model, uncle nodes have different weights depending on the child node been evaluated. Occupied uncle nodes sharing a face with the current child have weight 2, and the further away uncle nodes have weight 1. Empty nodes were not considered this time. For a child node to be set as occupied, a score of at least 6 should be observed.

These approaches led to results comparable to the smoothed NNI upsampling when looking at objective point-based metrics. However, the score-based SR methods developed are perceptually worse than the NNI upsampling because of the appearance of holes in the point cloud.

3.4 FRACTIONAL SUPER-RESOLUTION OF VOXELIZED POINT CLOUDS

A method for super-resolving voxelized point clouds using neighborhood inheritance from other frames was proposed by Garcia *et. al* proposed [5]. The idea was to super-resolve the current frame, using a dictionary of child nodes based on the neighborhood configuration from previous non-downsampled frames. We took this method, which worked only for the inter-frame case with exact level downsampling, and adapted it for the intra-frame case, also considering fractional downsampling.

We define the neighborhood state of a voxel $\varphi_M(\mathbf{v}(k))$ as an (M^3-1)-binary number indicating the occupancy of neighbor voxels inside an $M \times M \times M$ cube. Similarly, the child occupancy state of a parent voxel $\sigma(\mathbf{v}_d(k))$, is a $\lceil s \rceil^3$-binary number indicating which of the child voxels in $\mathcal{V}_u(k)$ are indeed occupied. The proposed method assumes that there are self-similarities at different scales of a point cloud geometry. We take the input point cloud geometry V_d and its correspondent down-

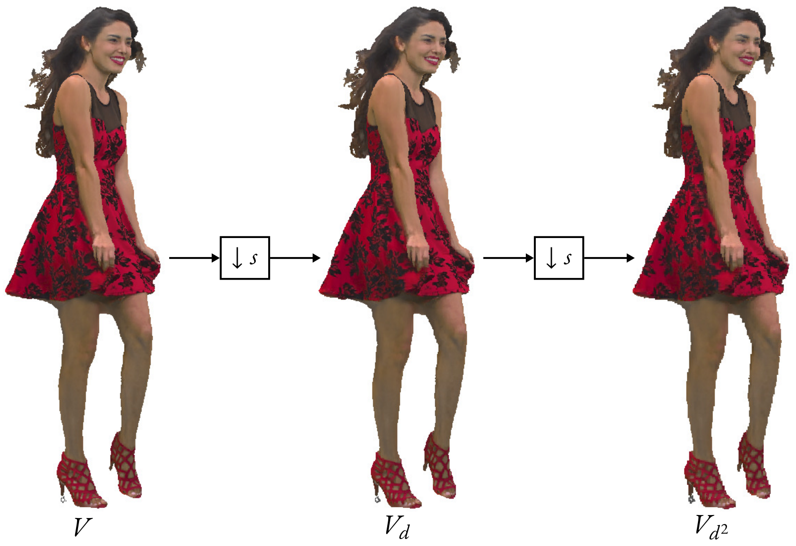


Figure 3.4: Illustration of the inputs utilized in the proposed SR method. By performing yet another downsampling in the input geometry V_d, we can "learn on-the-fly" how to super-resolve from V_{d^2} to V_d. Then, we use this knowledge to super-resolve from V_d back to V.

sampling factor s. Then, yet another downsampling is performed in the input geometry to generate V_{d^2}, as depicted in Figure 3.4. Using the input's parent voxels, we can check the relation of parent neighborhood occupancy states with child occupancy states and build a table, as illustrated in Table 3.2.

Table 3.2: Gathering data from the input point cloud.

Parent neighborhood state	Child occupancy state
$\varphi_M(\mathbf{v}_{d^2}(1))$	$\sigma(\mathbf{v}_{d^2}(1))$
$\varphi_M(\mathbf{v}_{d^2}(2))$	$\sigma(\mathbf{v}_{d^2}(2))$
$\vdots$	$\vdots$
$\varphi_M(\mathbf{v}_{d^2}(K))$	$\sigma(\mathbf{v}_{d^2}(K))$

To create the m-th entry of the dictionary, $m = 0, 1, \ldots, 2^{M^3-1} - 1$, we estimate the most likely child occupancy state for each possible value of $\varphi_M(m)$, i.e.,

$$\bar{\sigma}(m) = E\{\sigma(\mathbf{v}_{d^2}(k)) \mid \varphi_M(m)\}. \tag{3.9}$$

Neighborhood states not present in the input data are associated with fully occupied child states. The resulting dictionary represents a look-up-table (LUT), as illustrated in Table 3.3. Therefore, to super-resolve the input geometry, we compute its neighborhood states $\varphi_M(\mathbf{v}_d(k))$ and associate each value with the correspondent child occupancy using the LUT, i.e.,

$$\sigma(\mathbf{v}_d(k)) = \bar{\sigma}(m) \mid \varphi_M(m) = \varphi_M(\mathbf{v}_d(k)). \tag{3.10}$$

Since the LUT is indexed by the neighborhood states, there is no need to store $\varphi_M(m)$ in the LUT, so $\bar{\sigma}(m) = \mathrm{LUT}(m)$, and we can rewrite (3.10) as

$$\sigma(\mathbf{v}_d(k)) = \mathrm{LUT}\big(\varphi_M(\mathbf{v}_d(k))\big). \tag{3.11}$$

Thus, the set of super-resolved children from $\mathbf{v}_d(k)$ is

$$\mathcal{V}_{sr}(k) = \mathcal{V}_u(k \mid \sigma(\mathbf{v}_d(k))). \tag{3.12}$$

And finally, the super-resolved geometry,

$$V_{sr} = \bigcup_{k=1}^{K} \mathcal{V}_{sr}(k). \tag{3.13}$$

Table 3.3: Illustration of the dictionary used in the proposed SR method.

m	φ_M	$\bar{\sigma}$
0	`0000000...0000`	`1111 1111`
1	`0000000...0001`	`0000 1100`
$\vdots$	$\vdots$	$\vdots$
$2^{M^3-1} - 1$	`1111111...1111`	`1111 1111`

While the aforementioned outlines the basic ideas of the method's operation, some constraints must be defined, and some additional steps are introduced to improve the reconstructed geometry and to allow for fractional SR.

To get a symmetric neighborhood around a voxel, the neighborhood size, M, must be an odd number. As M increases by a single step, from 3 to 5, the number of dictionary entries surges from 2^{26} to 2^{124}. This makes using large values of M impractical, not only because it takes more computational effort to find a bigger neighborhood but also because building an effective dictionary with so many entries from a single frame is not possible. Considering that most of the entries would not be found in the data, the output geometry would be approximately equal to the NNI upsampling. This happens because the dictionary entries become overly specific. For this reason, we fixed the neighborhood size to its minimum, $M = 3$. To simplify notation, we drop the subscript of φ hereon.

The scale factor s must also be constrained because the dictionary's large size imposes memory restrictions. Each extra bit in $\bar{\sigma}(m)$ means 2^{26} additional bits to represent the dictionary. Moreover, as s increases, the number of meaningful entries in the dictionary decreases, since there is not much information in the lower levels of the geometry, as it is effectively becoming a single cube. Thus, we decided to constrain the values of s, $\{s \in \mathbb{Q} \mid 1 < s \leq 2\}$. Inside this interval, we can profit from partial downsampling and super-resolve a full octree level. If $s > 2$ is required, the SR method can still be used, by performing $t = \lceil \log_2(s) \rceil$ nested SRs with a new scale factor $s' \approx \sqrt[t]{s}$, where the approximation sign is needed since s' must be a fractional number.

A classification step is added before the LUT creation and before super-resolving the point cloud to ensure that the downsampling irregularities are taken into account. This way, seven different LUTs are created, each of them considering only labeled voxels with the specific parenthood conditions of Figure 3.1. Observing that parent voxels in classification 0 can be directly upsampled without error.

Finally, to improve the LUT quality using a single frame, we explore the fact that grid downsampling produces different results depending on the point cloud's position inside the bounding cube. Thus, we apply incremental shifts, $\Delta_{b:e}$, to the input geometry as a means of increasing input data to populate the dictionary better. We define $\Delta_{b:e}$, as the set containing all the the points inside the cube $[b, e]^3$. For example, when $b = 0$ and $e = 1$, $\Delta_{0:1}$ defines eight incremental displacements,

$$\Delta_{0:1} = \{(0, 0, 0), (0, 0, 1), (0, 1, 0), \ldots, (1, 1, 1)\}.$$

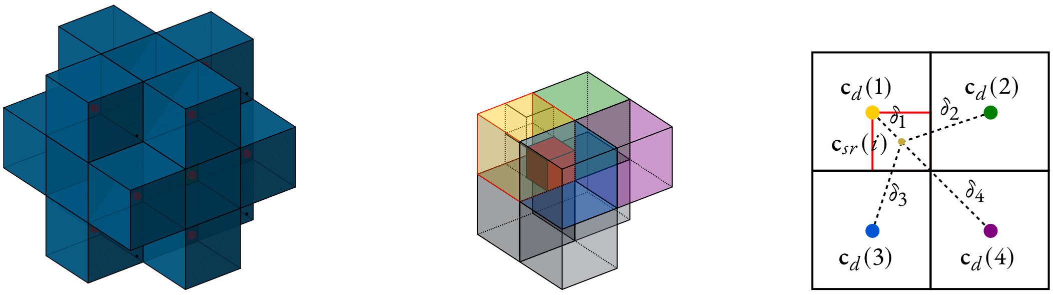
Experimentally, we found out that using these incremental shifts can considerably improve the super-resolved geometry. Using $\Delta_{-1:1}$, for example, improved the D1 metric of the super-resolved geometry at an average of about 2dB when comparing with SR without the displacements. However, larger displacements improved the geometry only marginally, and at a greater cost of time. If speed is a concern, then using $\Delta_{0:1}$ still improves the output at about 1.5dB on average, while being considerably faster than $\Delta_{-1:1}$.

To super-resolve texture, we borrow the color prediction idea used in TMC13 [62]. In the transform domain prediction of RAHT [75], the estimated color for each occupied child node is the average the parent node's color with the colors of the uncle nodes that share an edge with that child node, weighted by the inverse distance between each involved parent node and the current child node being estimated. We refer to this method as the weighted average of adjacent neighbors (WAAN). A variable weight, dependent on s, was introduced in the WAAN to take into account the fact that the parent color should be more important as the scale factor decreases. An example of the neighbors considered to estimate a child node's color is illustrated in Figure 3.5.

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

$$\mathbf{c}_{sr}(i) = \frac{\mathbf{c}_d(k) + \zeta \sum_{\ell} \delta_{\ell}^{-1} \mathbf{c}_d(\ell)}{1 + \zeta \sum_{\ell} \delta_{\ell}^{-1}}, \tag{3.14}$$

where $\zeta = \delta_1 s/8$, and ℓ represents the index of the occupied neighbors that share a face with $\mathbf{v}_{sr}(i)$, like illustrated in Figure 3.5(c).



(a) (b) (c)

Figure 3.5: Illustration of the neighbors used in the WAAN calculation. In (a), all the neighbors that must be considered in the 3 × 3 × 3 are shown. In (b), only the uncle nodes sharing a face with the highlighted child node are depicted. In (c), the illustration of how the distances are calculated. Only uncle nodes from the top were shown to ease representation, but the three uncle nodes from the bottom should also be considered in the calculation.

We synthesize the SR method using the following pseudo-code algorithms.

Algorithm 1: Intra-frame geometry SR by LUT

```
Input: $V_d$, $C_d$, $s$
Output: $V_{sr}$, $C_{sr}$

Build $\{\mathrm{LUT}_j(\varphi)\}$, using $V_d$ and $s$ as inputs, for the seven parenthood conditions;   // Algorithm 2
for ($k = 1$; $k \leq K$; $k = k + 1$) do
    $j = 4\wp(x_k) + 2\wp(y_k) + \wp(z_k)$;   // $\wp(x_k)$ is 1 if $x_k$ is multiparous, and 0 otherwise
    /* j=0 -> 1 child, xyz are uniparous                                   */
    /* j=1 -> 2 children, yz coordinates are uniparous                     */
    /* j=2 -> 2 children, xz coordinates are uniparous                     */
    /* j=3 -> 2 children, xy coordinates are uniparous                     */
    /* j=4 -> 4 children, only x coordinate is uniparous                   */
    /* j=5 -> 4 children, only y coordinate is uniparous                   */
    /* j=6 -> 4 children, only z coordinate is uniparous                   */
    /* j=7 -> 8 children, xyz are multiparous                              */
    if ($j = 0$) then
        $\mathcal{V}_{sr}(k) = \mathrm{round}(s \cdot \mathbf{v}_d(k))$;
        $\mathcal{C}_{sr}(k) = \mathbf{c}_d(k)$;
    else
        Get $\varphi(\mathbf{v}_d(k))$;
        $\sigma(\mathbf{v}_d(k)) = \mathrm{LUT}_j(\varphi(\mathbf{v}_d(k)))$;
        $\mathcal{V}_{sr}(k) = \mathcal{V}_u(k \mid \sigma(\mathbf{v}_d(k)))$;
        for ($i = 1$; $i \leq i_{\max} \mid \mathcal{V}_{sr}(k)$; $i = i + 1$) do
            $\zeta = \delta_1 s/8$;
            $\mathbf{c}_{sr}(i) = \frac{\mathbf{c}_d(k)+\zeta \sum_{\ell} \delta_l^{-1}\mathbf{c}_d(l)}{1+\zeta \sum_{l} \delta_l^{-1}}$;   // $\ell$ is such that $\mathbf{v}_d(\ell)$ shares a face with $\mathbf{v}_{sr}(i)$
            $\mathcal{C}_{sr}(k) = \{\mathcal{C}_{sr}(k); \mathbf{c}_{sr}(i)\}$;
$V_{sr} = \bigcup_{k=1}^{K} \mathcal{V}_{sr}(k)$;
$C_{sr} = \bigcup_{k=1}^{K} \mathcal{C}_{sr}(k)$;
```

Algorithm 2: LUT build-up

```
Input: $V_d, s$
Output: $\{\mathrm{LUT}_j(\varphi)\}$, are sets of lists linking a neighborhood state, $\varphi$, to a child state, $\sigma$, classified
        by $j \mid 1 \le j \le 7$, representing each parenthood condition

for ($m = 0;\ m \le 2^{26} - 1;\ m = m + 1$) do
    /* initialize each LUT with maximum children occupancy for each parenthood condition */
    $\mathrm{LUT}_1(m)$ = 0001 0001;        // 2 children, yz coordinates are uniparous
    $\mathrm{LUT}_2(m)$ = 0000 0101;        // 2 children, xz coordinates are uniparous
    $\mathrm{LUT}_3(m)$ = 0000 0011;        // 2 children, xy coordinates are uniparous
    $\mathrm{LUT}_4(m)$ = 0000 1111;        // 4 children, only x coordinate is uniparous
    $\mathrm{LUT}_5(m)$ = 1100 1100;        // 4 children, only y coordinate is uniparous
    $\mathrm{LUT}_6(m)$ = 0101 0101;        // 4 children, only z coordinate is uniparous
    $\mathrm{LUT}_7(m)$ = 1111 1111;        // 8 children, xyz are multiparous
for ($i = 1;\ i \le \mathtt{length}(\Delta_{b:e});\ i = i + 1$) do
    $V_d^{(i)} = V_d + \Delta_{b:e}(i) = \{\mathbf{v}_d^{(i)}(k)\}$;
    $V_{d^2}^{(i)} = \mathtt{unique}\left(\mathtt{round}(V_d^{(i)}/s)\right) = \{\mathbf{v}_{d^2}^{(i)}(\hat{k})\}$;
    for ($\hat{k} = 1;\ \hat{k} \le \hat{K}_i;\ \hat{k} = \hat{k} + 1$) do
        $j = 4\wp(x_{\hat{k}}) + 2\wp(y_{\hat{k}}) + \wp(z_{\hat{k}})$;        // $\wp(x_k)$ is 1 if $x_k$ is multiparous, and 0 otherwise
        Get $\varphi\left(\mathbf{v}_{d^2}^{(i)}(\hat{k})\right)$;
        Get $\sigma\left(\mathbf{v}_{d^2}^{(i)}(\hat{k})\right)$;        // this is done by comparing which points of $V_u^{(i)}$ exist in $V_d^{(i)}$
        $\mathtt{table}_j = \left[\mathtt{table}_j; \varphi\left(\mathbf{v}_{d^2}^{(i)}(\hat{k})\right), \sigma\left(\mathbf{v}_{d^2}^{(i)}(\hat{k})\right)\right]$;        // gathering data for each condition
for ($j = 1;\ j \le 7;\ j = j + 1$) do
    for ($m = 0;\ m \le 2^{26} - 1;\ m = m + 1$) do
        if ($\exists\, m \in \mathtt{table}_j[1]$) then
            $\mathrm{LUT}_j(m) = E\{\mathtt{table}_j[2] \mid \mathtt{table}_j[1] = m\}$;
```

3.5 SUBJECTIVE QUALITY ASSESSMENT APPLICATION

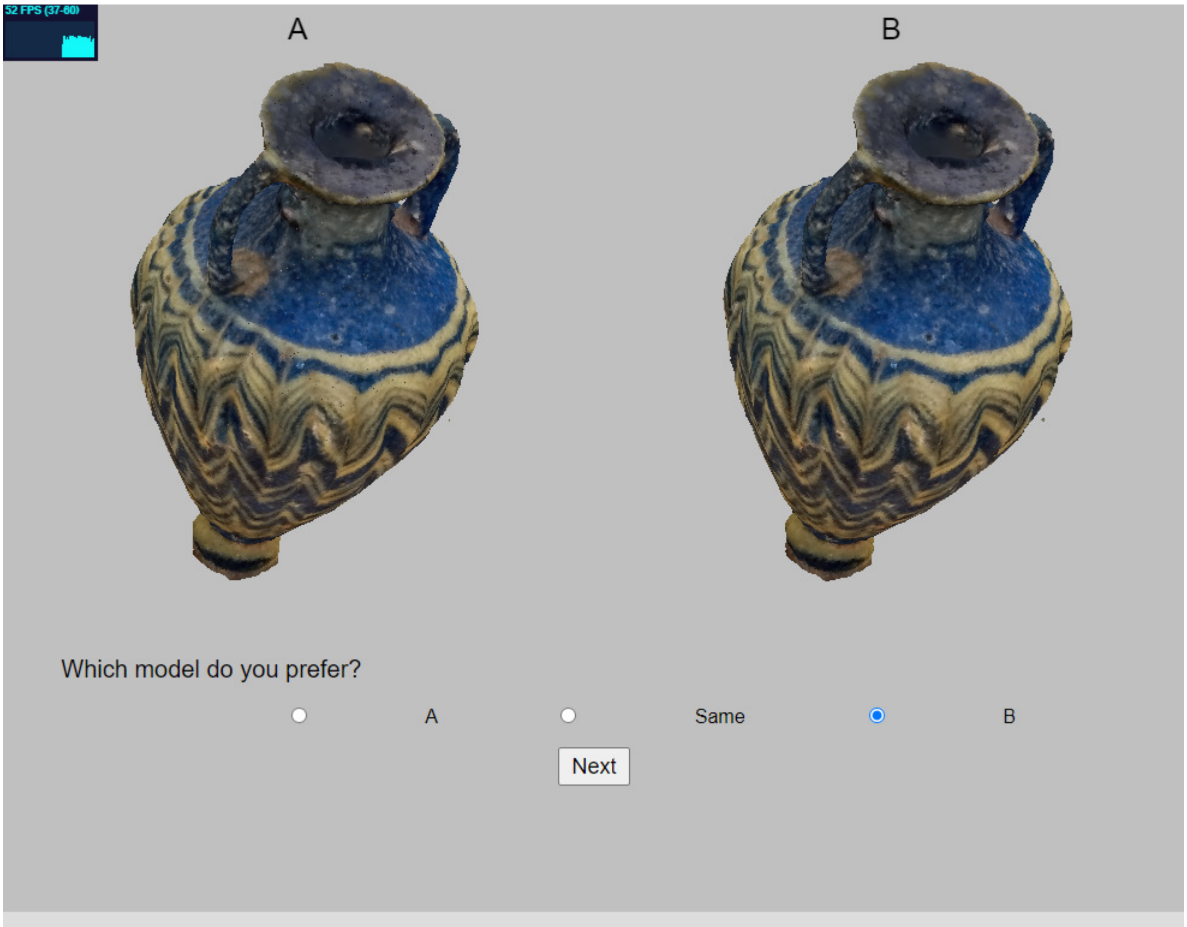
As briefly discussed in Section 2.4, evaluating distortion on point clouds is not straightforward, and objective metrics may, sometimes, lead to misleading results. In light of these issues, we planned to assess distortions introduced by the SR methods in a subjective evaluation using the renderer developed in collaboration with the École Polytechnique Féderale de Lausanne (EPFL) [6]. However, the experiment was not carried out due to the restrictions on lab access imposed by the COVID-19. Although an internet-based crowdsourcing experiment was an option, as we waited for the pandemic situation to improve, the time set for experiment completion became an issue, and we opted not to proceed with the subjective evaluation. Nonetheless, in the following Section, we present the developed renderer, which is ready to be used in a future subjective assessment evaluation.

3.5.1 Web-based Renderer

We developed an interactive renderer in JavaScript using the `three.js` library [105], which has built-in support for loading point cloud in both `PLY` and `PCD` file formats. The loaded geometry is converted to the `Points` class, which displays voxels using point primitives for the 2D RBF. By default, point primitives are squares, and the user may change its shape by importing a custom RBF geometry using an image file. The `PointsMaterial` class defines the color, which is loaded by the input point cloud file, and the size of the points, which can be configured to be either fixed or adaptive. Point size information needs to be calculated, and we opted to use off-line calculations to reduce the computation overhead of the rendering software. So to render a point cloud in our software, the user must input a `PLY` or `PCD` file containing the geometry and the color of the desired point cloud, and a `JSON` configuration file.

A renderer application in `three.js` requires a virtual scene, a camera, and a renderer with an associated canvas (a projection plane). We start by creating a virtual scene with the point cloud model placed in the middle. The background color can be set by the user in the configuration file. For assessment purposes, we chose a mid-gray value. The scene is captured using an orthographic camera. To enable interactivity and provide 3D depth cues, we opted to use a camera with trackball control, enabling the user to change the camera parameters using mouse movements. Those changes are updated at a fast rate (over 30fps in a wide variety of devices and up to 60fps on high-end ones), ensuring smooth transitions of viewpoints for a better immersive experience. The image captured by the camera is rendered to a canvas using a `WebGLRenderer` object. The canvas dimensions can be either specified using the configuration file or adaptive to the window size. A snapshot from the application is shown in Figure 3.6.

There are some options to define the initial point size value. Since the splats are always parallel to the view plane, one-to-one representation from voxels to pixel leads to holes in the perceived surface when the camera is rotated (Figure 2.9.) To enable watertight surfaces from different viewpoints, increasing the point size is necessary. Point size can be assigned either globally, using the intrinsic resolution, or locally, using each point's nearest neighbor distance. The initial point size is then



Which model do you prefer?

○ A ○ Same ● B

Figure 3.6: Illustration of the renderer application. For nearly equal stimuli, a pair comparison approach is employed using a ternary voting system.

adjusted based on the camera zoom parameter. An additional scaling factor is provided as a global compensating quantity to be adjusted depending on the sparsity of the model and the desired visual results [6].

The software can also record the user's interactivity information to see which viewpoints are most relevant in each content. This feature is useful for providing weights to the projection-based metric's averages, using the time spent at each viewpoint to increase its importance in the calculation.

4 RESULTS AND DISCUSSION

4.1 EVALUATION FRAMEWORK

The presented SR methods from the previous Chapter were all attempts to improve the NNI upsampling. Thus, considering that, as far as we know, at the time of this writing, there are no other methods to super-resolve grid downsampled voxelized point clouds in the literature, we decided that the best way to assess the quality of the proposed LUT-based SR method was to compare it with the NNI upsampling. To mitigate the aliasing effects from the base approach, we also considered the smoothed NNI upsampling.

We start by presenting the chosen datasets and test conditions. Then the results are presented and discussed. In the final Section of this Chapter, we present a direct application for the LUT-based SR method for PCC.

4.2 DATASETS AND TEST CONDITIONS

Considering that the proposed SR method was developed with intra-prediction in mind, we focused on category 1 contents (static objects and scenes) with less than 4 million voxels, due to the current version of the code's memory restrictions.

The proposed method can be largely affected by the point cloud capturing and voxelization processes due to its statistical nature. For example, if the capturing process adds a thick layer of hidden points to the point cloud, its appearance may not change, but it can significantly impact the neighborhood statistics and, consequently, the creation of the LUT. To perform the experiment, we prioritize point clouds from the CTC [85] which are originally voxelized, avoiding biases introduced in the voxelization process. Nonetheless, there are just a few of those point clouds, so we also needed to add point clouds that required a pre-processing voxelization step.

To reduce the number of comparisons and to obtain a more representative result, we clustered the point clouds sharing the same source, voxel depth, and density into groups, whenever possible.

In order to evaluate the sparsity of a point cloud, we define the density measure ρ_φ as the average neighborhood occupancy-rate of adjacent voxels to an occupied voxel. This way, a value of $\rho_\varphi = 0.5$ indicates that each voxel of the point cloud has, on average, 13 occupied neighbors of the 26 in the $3 \times 3 \times 3$ neighborhood. Empirically, we found out that point clouds in which $\rho_\varphi \geq 0.3$ presented projections with a one-to-one relationship between rendered pixels and voxels without holes (watertight projections). This represents an average of about 8 neighbors per occupied voxel. We consider point clouds in this situation to be *dense*. If, however, $0 < \rho_\varphi < 0.3$ the point cloud is considered *sparse*. If $\rho_\varphi = 0$ for a given neighborhood size M, but $\rho_\varphi > 0$ for a neighborhood size

Table 4.1: Summary of information of the point clouds representing human figures.

Content	Source	Voxelization	Voxel depth	# voxels	ρ_φ
Group: *8i_vox10*	8i				
longdress		✗	10-bit	857,966	0.429
loot		✗	10-bit	805,285	0.428
redandblack		✗	10-bit	757,691	0.433
soldier		✗	10-bit	1,089,091	0.432
Group: *8i_vox12*	8i				
boxer		✗	12-bit	3,493,085	0.031
longdress12		✗	12-bit	3,096,122	0.027
loot12		✗	12-bit	3,017,285	0.029
redandblack12		✗	12-bit	2,770,567	0.025
thaidancer	8i	✗	12-bit	3,130,215	0.332
Group: *owlii*	Owlii				
basketball_player		✗	11-bit	2,925,514	0.452
dancer		✗	11-bit	2,592,758	0.445
queen	Technicolor	✗	10-bit	1,000,993	0.524
Group: *MVUB*	Microsoft				
andrew9		✗	9-bit	279,664	0.547
david9		✗	9-bit	330,797	0.542
phil9		✗	9-bit	370,798	0.543
ricardo9		✗	9-bit	214,656	0.550
sarah9		✗	9-bit	302,437	0.538

$M' > M$, then the point cloud is considered *regularly-spaced dense*, and it is possible to reduce an initially assumed sparse point cloud into a dense one at a smaller resolution. This typically occurs either after an inefficient voxelization process, or when the upsampling by simple expansion is done in an LR point cloud, such as in the lossy octree configuration used in the TMC13 codec [62].

Tables 4.1 and 4.2 summarize information about the chosen contents, while Figure 4.1 depicts representatives viewpoints form each group.

Point clouds representing human figures were captured with more recent technology and in a controlled environment, ensuring a better quality to them. The *8i_vox10* group [50] is comprised of *longdress* (longdress_vox10_1300), *loot* (loot_vox10_1200), *redandblack* (redandblack_vox10_1550), and *soldier* (soldier_vox10_0690). The *8i_vox12* group [106] is comprised of *boxer* (boxer_viewdep_vox12), *longdress12* (longdress_viewdep_vox12), *loot12* (loot_viewdep_vox12), and *redandblack12* (redandblack_viewdep_vox12); soldier_viewdep_vox12 was not used because it surpasses the voxel cap. *Thaidancer* (Thaidancer_viewdep_vox12) [106], although sharing the same source of the rest of the *8i_vox12* group, has different characteristics from the others, as can be ob-

Table 4.2: Summary of information of the point clouds representing objects.

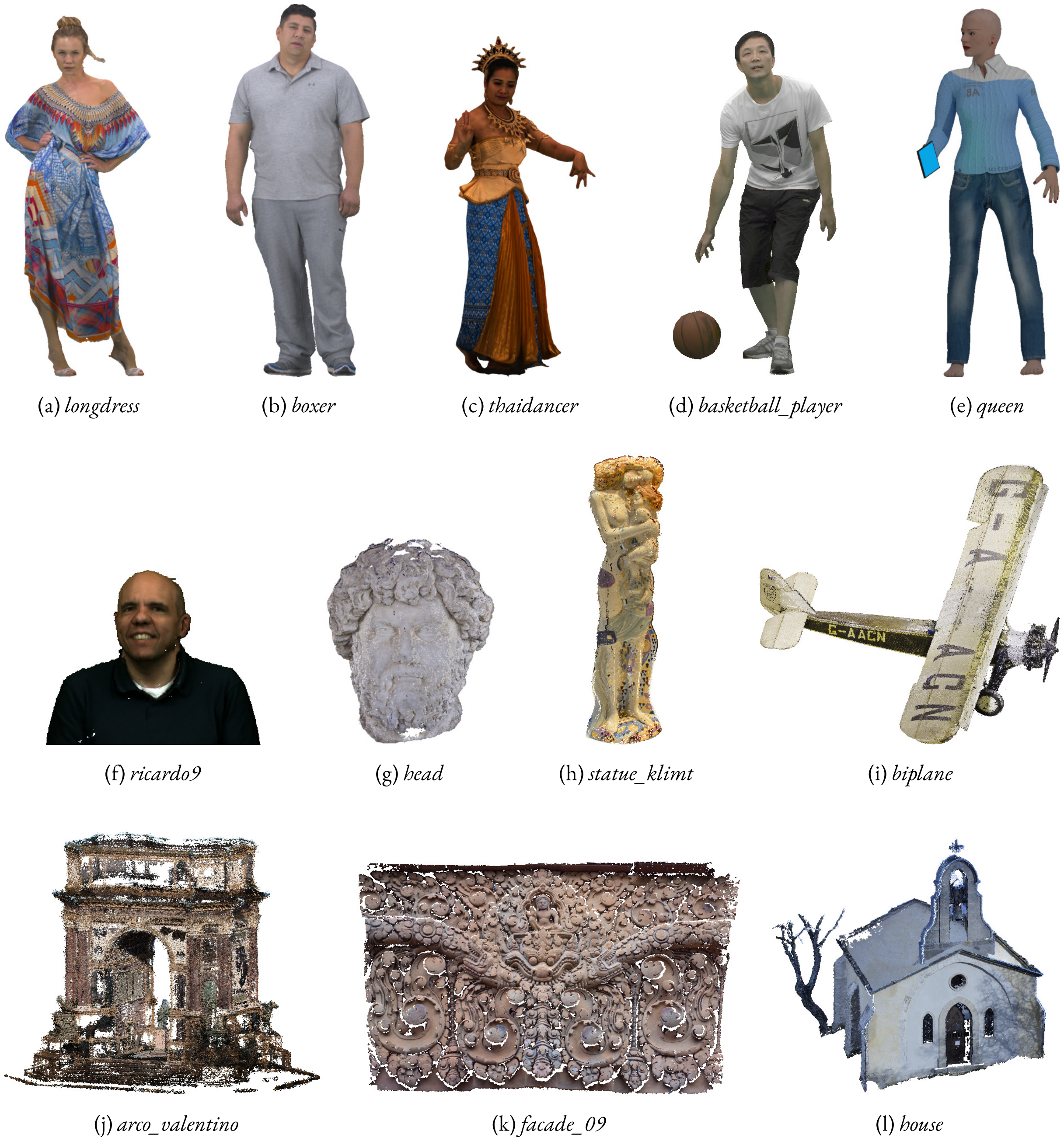
Content	Source	Voxelization	Voxel depth	# voxels	ρ_φ
head	MPEG	✓	9-bit	938,112	0.532
biplane	ScanLAB	✓	10-bit	1,181,016	0.567
statue_klimt	MPEG	✓	10-bit	483,068	0.209
arco_valentino	UPM	✗	12-bit	1,481,746	0.025
facade_09	MPEG	✓	11-bit	1,560,786	0.165
house	MPEG	✓	10-bit	3,638,139	0.247

served by its density, and for that it was separated. The *owlii* group [14] contains *basketball_player* (basketball_player_vox11_00000200) and *dancer* (dancer_vox11_00000001). A synthetic human figure is represented in *queen* (queen_frame_0200) from Technicolor. The Microsoft Voxelized Upper Body (MVUB) group [107] contains the first frame of each of the sequences of *andrew9*, *david9*, *phil9*, *ricardo9*, and *sarah9*. Contents from *8i_vox10*, *8i_vox12*, *thaidancer*, *owlii*, and *queen* can be found in the MPEG repository[1], while *MVUB* can be found in the JPEG Pleno repository[2].

On the other hand, point clouds representing objects were captured using older technologies, and most of them were originally meshes that needed to be voxelized. Many point clouds in this category have more than 4 million occupied voxels, so we had to downsample some of them to reach the point cap. This was the case for *head*, whose original version found in the MPEG[1] (Head_00039_vox12) has over 13 million points, and *house* (House_without_roof_00057_vox12) which originally has over 4 million points. Although under the point cap, other point clouds were extremely sparse, but not quite regularly-space dense, $\rho_\varphi \gtrapprox 0$. Because of that, downsampling those point clouds with a factor $s = 2$ would decimate less than 1% of the original points, making SR unjustified. In these cases, we reduced the bit depth to increase density. This was done for *statue_klimt* (Statue_Klimt_vox12) and *facade_09* (Facade_00009_vox20), which can also be found in the MPEG repository[1]. *Arco_valentino* was originally a mesh captured by the Universidad Politécnica de Madrid (UPM), but a voxelized version can be found in the MPEG repository[1] (Arco_Valentino_Dense_vox12), and this was used without modifications. Lastly, *biplane* (1x1_Biplane_Combined_000) is also originally a mesh found in the JPEG Pleno repository[2], which needed to be voxelized for our purposes.

[1] https://mpegfs.int-evry.fr/mpegcontent/

[2] http://plenodb.jpeg.org/



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

(f) *ricardo9* (g) *head* (h) *statue_klimt* (i) *biplane*

(j) *arco_valentino* (k) *facade_09* (l) *house*

Figure 4.1: Representative viewpoints of some of the test models. Human figures are represented from (a) to (f), whilst objects are represented from (g) to (l).

4.3 ASSESSING THE QUALITY OF THE PROPOSED SUPER-RESOLUTION METHOD

Ten LR versions of each input point cloud were created by varying the scale factor s in the interval [1.1, 2]. Then, for each LR point cloud, we created an NNI upsampling version, a smoothed NNI upsampling version using the LS (NNI+LS), and an SR version using the algorithm of Section 3.4 (LUT). To assess each version's quality, we chose to use six metrics since point clouds assessment metrics are rather specific in the type of distortion they can capture. Point-based metrics are useful for capturing how close the two sets of compared points are. For that, we chose PSNR_{D1} (2.13), PSNR_{D2} (2.14), and PSNR_{Y} (2.18), which is referred to as Luma end-to-end PSNR.

Projection-based metrics are, in general, better to capture the visual fidelity of the distorted content. PPSNR (2.23), PSSIM (2.26), and PVIFP (2.29) were chosen for the projection-based assessment. Those metrics, however, are dependent on the selected rendering method. Intending to isolate the SR distortions, and since no subjective assessment was performed, we chose not to increase the point size in the renderer configurations. The idea was not to add further rendering distortions that could change the upsampling methods' intrinsic distortions. Thus, to get the projections, points were rendered as unit-sized cubes. Only six projection views were used (the six faces of the bounding cube containing the point cloud) to avoid aliasing of the other view angles. In this way, each visible voxel is projected into a single pixel so that in a 10-bit point cloud, there are six image projections of 1024×1024 pixels. The background color was set to a mid-gray value, and to make the metrics more sensitive, we only considered the rectangular region formed by the union between the foregrounds of the reference and the distorted projections.

The density measure ρ_φ is a good predictor of the proposed method's performance. Analyzing what happens with ρ_φ of a point cloud at different scales, we can illustrate its prediction property. In Figure 4.2, we use as an example a sparse and a dense point clouds to illustrate the density behavior with the downsampling. The sparse point cloud represented by *longdress12* has a steep slope in its density curve for $1 \leq s \leq 2$, indicating significant changes in the geometry structure occur when the downsampling is performed. The point cloud's original sparsity quickly vanishes as s increases, misleading the LUT formation from a denser point cloud. When operating in point clouds with flatter density slopes, as in *longdress*, the geometry structures of the downsampled versions are more similar to the original point cloud, so the algorithm has better chances of making the right choices about child occupancy.

Figures 4.3 and 4.4, depict the point-based metrics assessment for the human figures and for the objects models, respectively. Looking at D1 and D2 metrics on the first two columns of the graphics from both figures, we can see that the LUT method achieved superior results in every case for lower values of s. Noticing that, in such cases, there are less points that need to be super-resolved, than for larger values of s. For dense point clouds ($\rho_\varphi \geq 0.3$), the LUT method was better than both NNI and NNI+LS, by about 5dB and 2dB, respectively, for all values of s. In sparser models, however, the gain was smaller, and in some cases, the NNI+LS method was virtually tied or better than the LUT, notably for higher values of s. Remember that, to super-resolve V_d with a factor of s, we need to downsample V_d by s once more to build the LUT (Algorithm 2). Thus, we expect the geometry

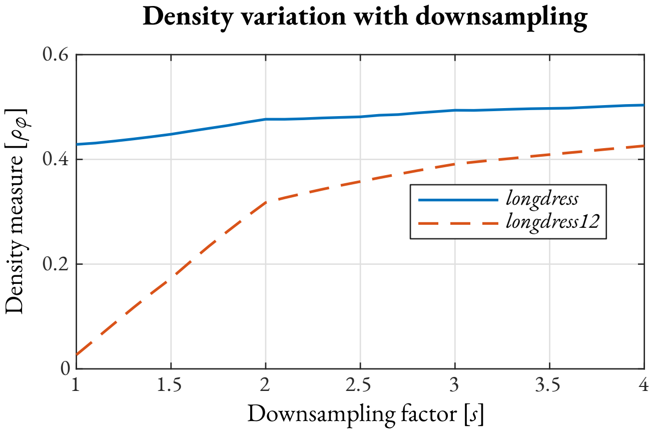


Figure 4.2: Comparison of the density variation with the downsampling factor between dense and sparse point clouds.

structure to be maintained somewhat unchanged by a downsampling factor of s^2. By comparing how ρ_φ changes with s in sparse and dense point clouds, we can see that it was expected for the LUT method to struggle when employed to super-resolve the former kind of content.

In the dense sets *queen*, *MVUB*, *head*, and *biplane*, LUT gains over the NNI+LS were not as significant as the others. All those sets have some form of bias in their geometry, like holes, spurious points, and thick hidden layers, which have proved detrimental to the LUT method. One interesting thing to notice is how consistent the geometry metrics were in the group sets (*8i_vox10*, *8i_vox12*, *owlii*, and *MVUB*), indicating that the capturing and processing methods are more relevant than the contents themselves for any of the upsampling methods.

The point-based texture assessment in the third column of graphics from Figures 4.3 and 4.4 also showed better results for the LUT method in almost every case. It is important to acknowledge that PSNR$_Y$ has a dependence on the geometry quality [108], so part of the gains in color is indeed related to the gains in the geometry quality. The degradation of texture in sparse sets in the NNI+LS method is somewhat interesting. The texture changes more abruptly among neighboring voxels in sparse point clouds than in dense ones. Thus, averaging neighboring textures in the former case is more prone to cause errors.

Differently from what happened with the geometry, the standard deviations of texture metrics in the group sets were quite prominent. This happened because there is a greater dependence on the content when upsampling color. For example, observe the difference in the texture of the shirt patterns displayed in Figure 4.5. Nonetheless, the trend is well captured by the mean in those cases.

Figures 4.6 and 4.7 show the projection-based metrics assessment for human figures and for objects models, respectively. In the first column of graphics, we can see that the PPSNR results showed a good correlation with the point-based results, with the LUT method being the best overall. This was expected since all of those are distance-based metrics, and they rely on a similar calculation method. The low values of PPSNR observed, particularly for sparse point clouds, occur because of the way we chose to render the projections. The majority of errors made by NNI upsampling-related methods are usually of excess nature, i.e., they add more points than necessary in their at-

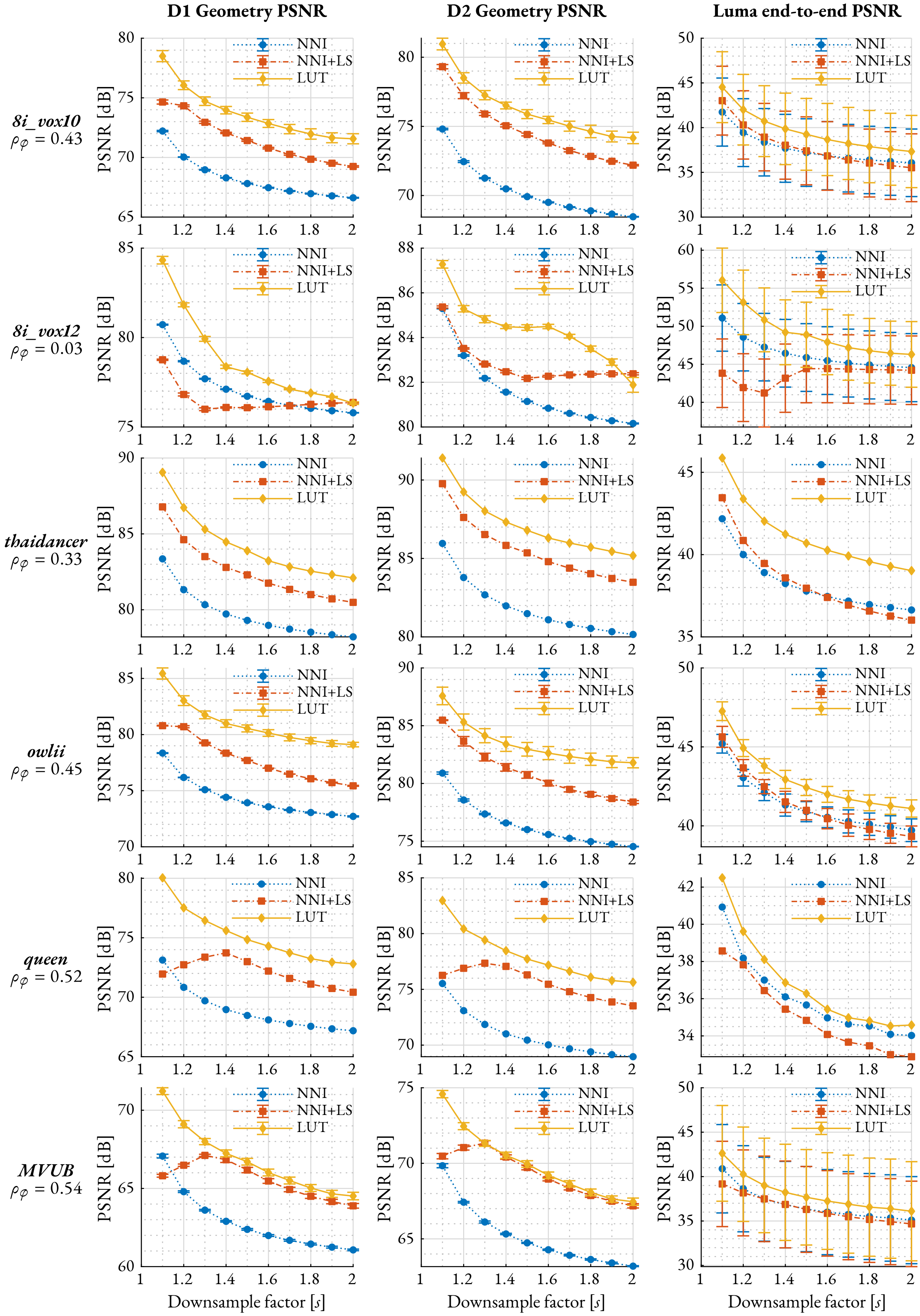


Figure 4.3: Point-based metrics for the point clouds representing human figures.

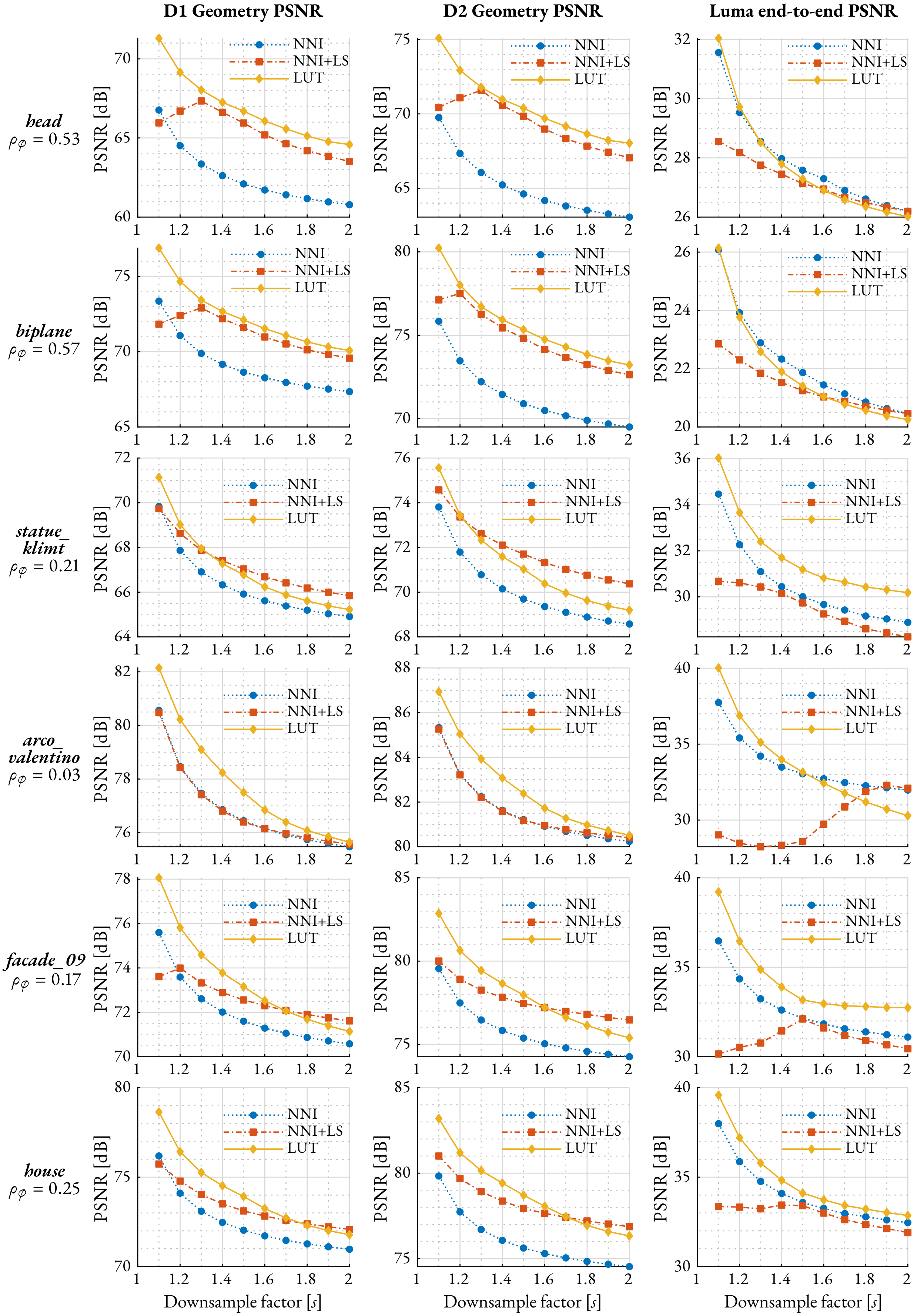
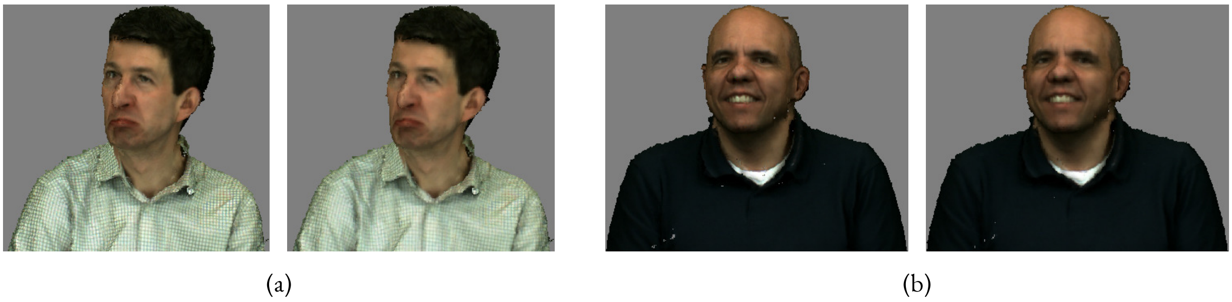


Figure 4.4: Point-based metrics for the point clouds representing objects.



(a) (b)

Figure 4.5: Texture comparison for upsampling. In both cases, we have the original point cloud on the left side, followed by the LUT SR version for $s = 1.5$ on the right side. The pattern in *andrew9*'s shirt in (a) is more challenging to be upsampled than the one in *ricardo9*'s shirt in (b). So we expect higher PSNR_Y values for *ricardo9* than for *andrew9*.

tempts to recreate the original point cloud. In sparse point clouds, those errors are more apparent, reducing the absolute level of the PPSNR. If a different rendering approach was used, probably higher values were to be observed.

Results from PVIFP and PSSIM do not share a high correlation with point-based metrics, however. Those metrics assess visual fidelity using aspects more related to the HVS than the PPSNR. Their results add a different perspective to each method's performance, which is greatly dependent on the rendering choice. PVIFP results, on the second column of graphics in Figures 4.6 and 4.7, were more dependent on each content. In the human figures set, the LUT and the NNI+LS methods outperformed the NNI upsampling, with the former having better results at lower values of s, and the latter gaining for higher downsampling factors. The NNI upsampling method was usually rated as the best method for the objects set, with a preference for the NNI+LS version at higher values of s. PSSIM results, on the third column, although not as strongly correlated with the point-based metrics as the PPSNR, showed a slight preference for the LUT method, especially for the human figures.

Although not a unanimous consensus among all the six assessed metrics, we can see a preference for the LUT method, notably for denser point clouds. It should be pointed out that there is no perfect metric, and each of them should be seen as a fidelity assessment from a different point of view and considering different aspects. Withal, the current standard metrics for evaluating and guiding the evolution of compression and processing of point clouds are the point-based ones. Furthermore, seeing that the optimization of the point-based metrics guided the development of the proposed SR method, it is fair to say that the LUT method outperformed the alternatives by the presented results.

Be that as it may, we should underline that the current choice of using the maximum error as the final metric, i.e., $\mathrm{D1}^{\mathrm{MSE}} = \max(\mathrm{D1}_{\mathcal{O}}^{\mathrm{MSE}},\ \mathrm{D1}_{\mathcal{D}}^{\mathrm{MSE}})$ from (2.11), can lead to misleading results. As illustrated in Figures 2.24 and 2.25, the $\mathrm{D1}_{\mathcal{O}}^{\mathrm{MSE}}$ measurement relates to errors by the omission of correct points, incorrect by missing, whilst the $\mathrm{D1}_{\mathcal{D}}^{\mathrm{MSE}}$ measurement relates to errors by the excess of points, incorrect by addition. Shifts in the degraded geometry can be viewed as a combination of the two errors, a correct point is missed, while an extra incorrect point is added. When developing SR methods, we had to constantly check both measurements to see whether the method was adding

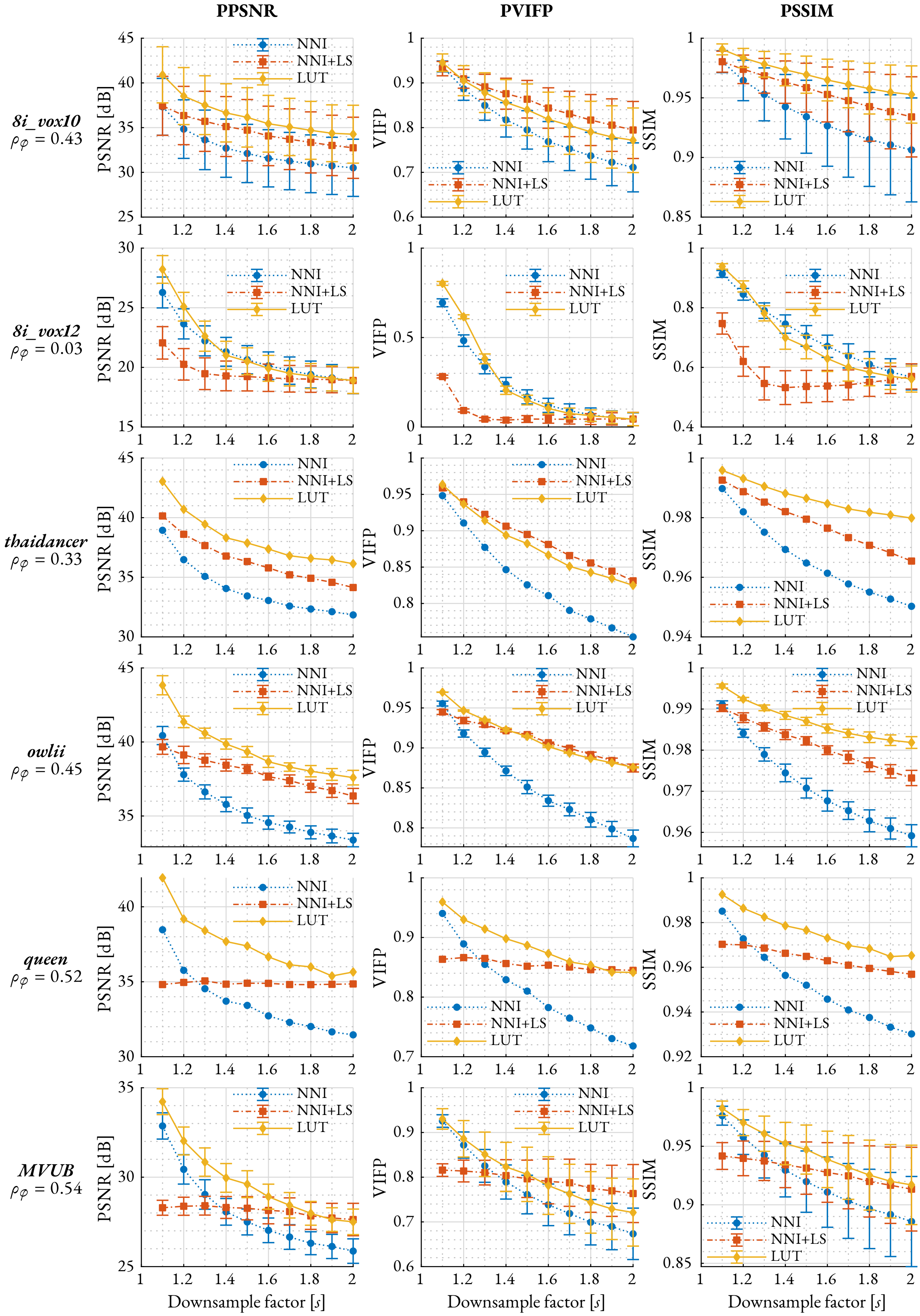


Figure 4.6: Projected-based metrics for the point clouds representing human figures.

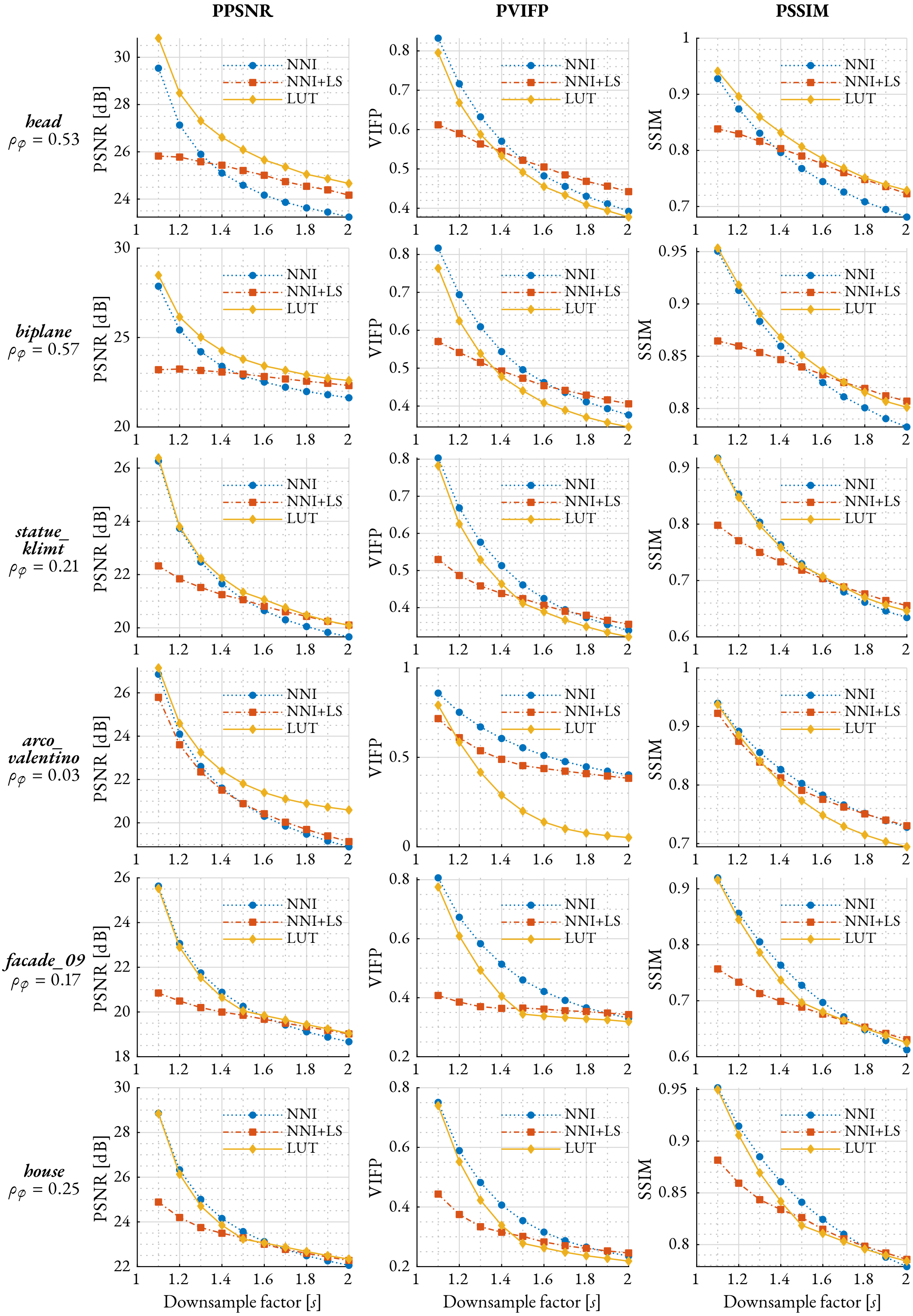


Figure 4.7: Projected-based metrics for the point clouds representing objects.

or removing too many points. When only one of these measurements is taken as the final metric, there is an underlying assumption that both measurements have the same behavior through all the distortion channel.

This problem can be illustrated by looking at the two measurements of the $\mathrm{D1}^{\mathrm{MSE}}$ metric from *arco_valentino*, Figure 4.8. We can see that the maximum error (excess error) for both NNI and NNI+LS methods is virtually the same, although omission errors are quite different. Using only the maximum error completely discards this information, and both methods are rated as having the same result, as depicted in Figure 4.4. If we consider the average value as the final D1 metric, the bottom right graphic of Figure 4.8, the results from the point-to-point metric become much more correlated to what was observed for *arco_valentino* in the PVIFP and PSSIM metrics.

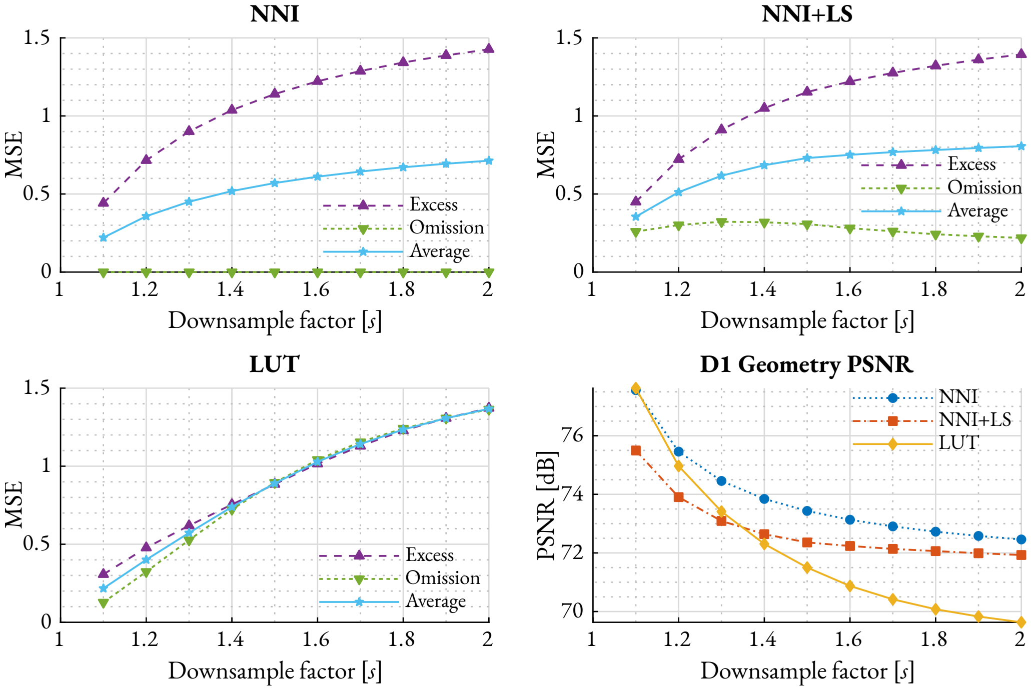
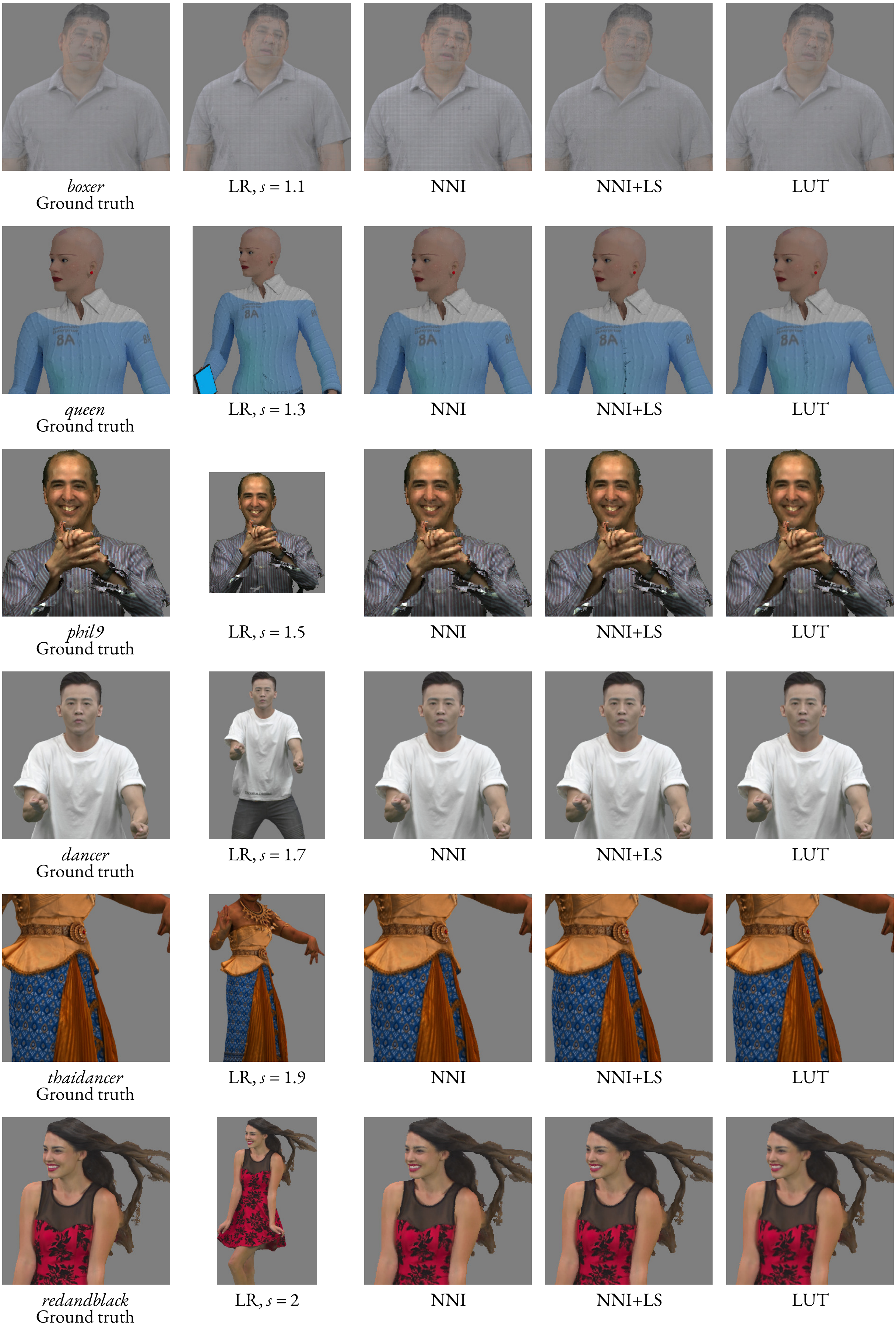


Figure 4.8: Behavior of point-to-point measurements for *arco_valentino*. Considering both error measurements, or perhaps their average value, can give a better understanding of the distortions present in each upsampling method, which is more accurate than only considering the maximum value. The three first plots show the behavior of the omission ($\mathrm{D1}_{\mathcal{O}}^{\mathrm{MSE}}$), excess ($\mathrm{D1}_{\mathcal{D}}^{\mathrm{MSE}}$), and average MSE for the NNI, NNI+LS, and LUT methods, respectively. The last plot shows how the final $\mathrm{PSNR}_{\mathrm{D1}}$ metric would be if the average MSE was considered.

Although this will not happen for every case, which is even illustrated by the unchanged behavior of the LUT curve in Figure 4.8, we believe that using both values of the point-based error measurements in the final metric may improve the metric correlation with subjective evaluation.

Viewpoints used in the projection-based metrics assessment for some of the point clouds are shown in Figures 4.9 and 4.10 for visual comparison. The full range of results for all tested point clouds can be found online[3].

[3] http://www.gitlab.com/tomasborges/fractional-sr-vox-point-clouds



boxer
Ground truth

LR, $s = 1.1$

NNI

NNI+LS

LUT

queen
Ground truth

LR, $s = 1.3$

NNI

NNI+LS

LUT

phil9
Ground truth

LR, $s = 1.5$

NNI

NNI+LS

LUT

dancer
Ground truth

LR, $s = 1.7$

NNI

NNI+LS

LUT

thaidancer
Ground truth

LR, $s = 1.9$

NNI

NNI+LS

LUT

redandblack
Ground truth

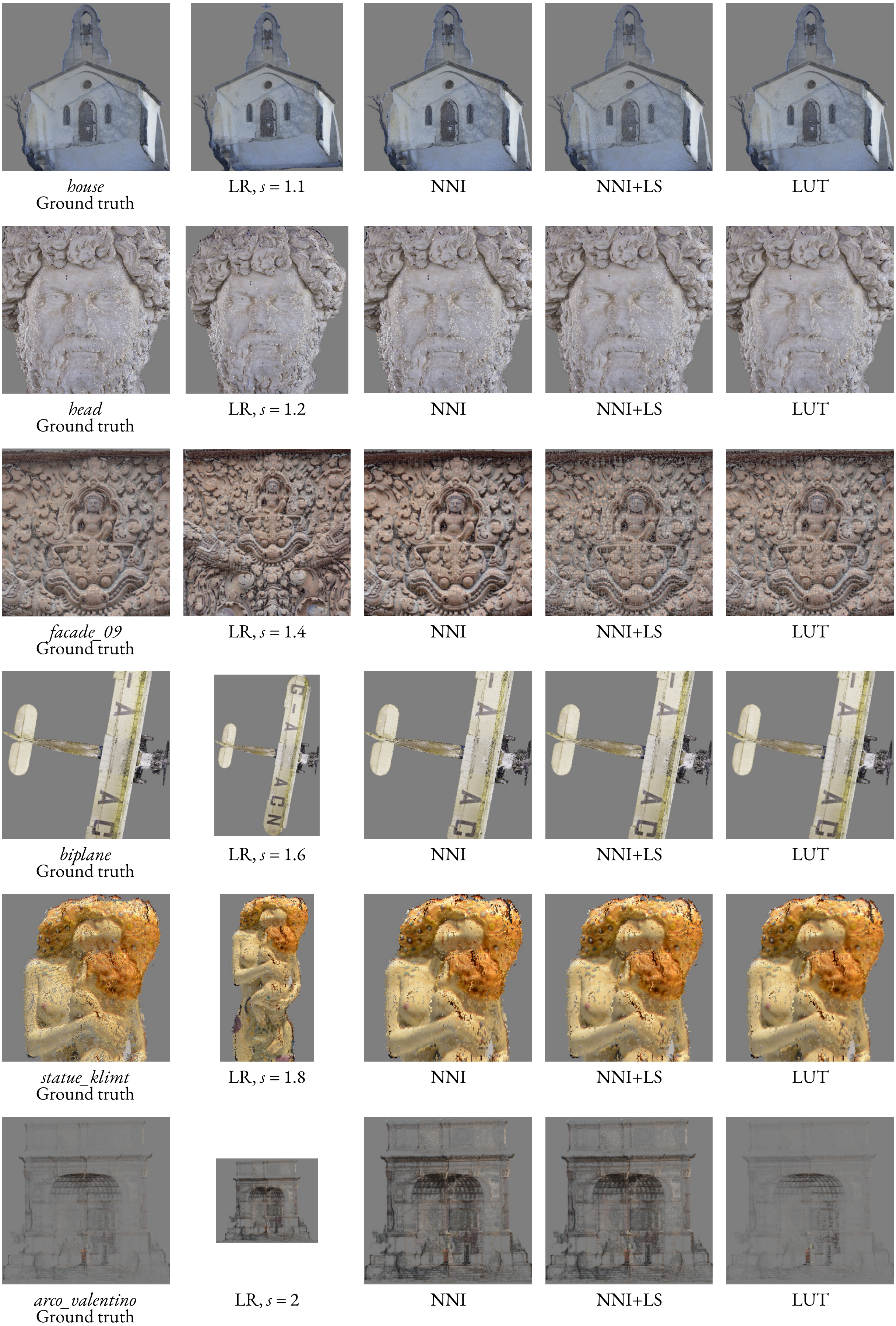
LR, $s = 2$

NNI

NNI+LS

LUT

Figure 4.9: Viewpoint projections for some of the point clouds with human figures.



house Ground truth | LR, $s = 1.1$ | NNI | NNI+LS | LUT

head Ground truth | LR, $s = 1.2$ | NNI | NNI+LS | LUT

facade_09 Ground truth | LR, $s = 1.4$ | NNI | NNI+LS | LUT

biplane Ground truth | LR, $s = 1.6$ | NNI | NNI+LS | LUT

statue_klimt Ground truth | LR, $s = 1.8$ | NNI | NNI+LS | LUT

arco_valentino Ground truth | LR, $s = 2$ | NNI | NNI+LS | LUT

Figure 4.10: Viewpoint projections for some of the point clouds with objects.

4.4 USING SUPER-RESOLUTION FOR INTERPOLATIVE COMPRESSION

The good performance of the LUT method observed in the last Section can be applied towards compression, be that for context generation or for interpolative compression. As we saw in Section 2.3.2, in the TMC13 codec, lossy geometry can be achieved either by just using a pruned octree or, when the *trisoup* method is enabled, by building a surface interpolation on top of the pruned octree. Thus, a direct application for interpolative compression can be made by super-resolving the output of G-PCC's pruned octree with our method at the decoder side. The approach is similar to what the *trisoup* does to improve the pruned octree geometry, but since it does not require the original point cloud, it can be done after the decoding step, avoiding sending extra bits for the segment indicators and vertex positions to refine the point cloud resolution. This approach's downside is that the recolorization step cannot be performed with the decoded point cloud as the original color information is not present. So we expect the texture from the *trisoup* method to be better, albeit at a higher bit-rate.

In TMC13, when lossy geometry is performed using only the pruned octree, the decompressed point cloud is upsampled at the decoder side without any interpolation. Only expansion is performed, $V_e = V_d \cdot s$. If our SR method is used at the decoder instead of the simple expansion, we can improve distortion results without changing the compression rate. The experiment was performed for *longdress*, *loot*, *redandblack*, and *soldier* comparing both octree and *trisoup* geometry encoders rate *versus* distortion plots (G-PCC version 8.0). For all cases, the color attributes were compressed using the RAHT encoder [67] with quantization steps determined by the MPEG protocol in the CTC [85]. For the first two lower rate points, where $s = 8$, and $s = 4$, we used the proposed algorithm in three ($2\times2\times2$) and two (2×2) steps, respectively, since at the current version of the code, the proposed method is limited to super-resolving a maximum scale factor of $s = 2$. In the last four rate points $s = 2$, $s = 4/3$, $s = 8/7$, and $s = 16/15$, the algorithm was applied just once. Even though our focus is not on color, our luma results are still very competitive, while the geometry easily outperforms the alternatives in all cases. The best results are achieved when s is inside the $[1, 2]$ interval. Results are depicted in Figure 4.11. When considering projection-based metrics, Figure 4.12, we also obtained superior results. In Figure 4.13, projections from *soldier* were gathered. It is remarkable to see the improvements in geometry delivered by the proposed method in dense point clouds. The absence of holes when compared to the *trisoup*, and the refined edges when compared to the pruned octree make for a much better output geometry.

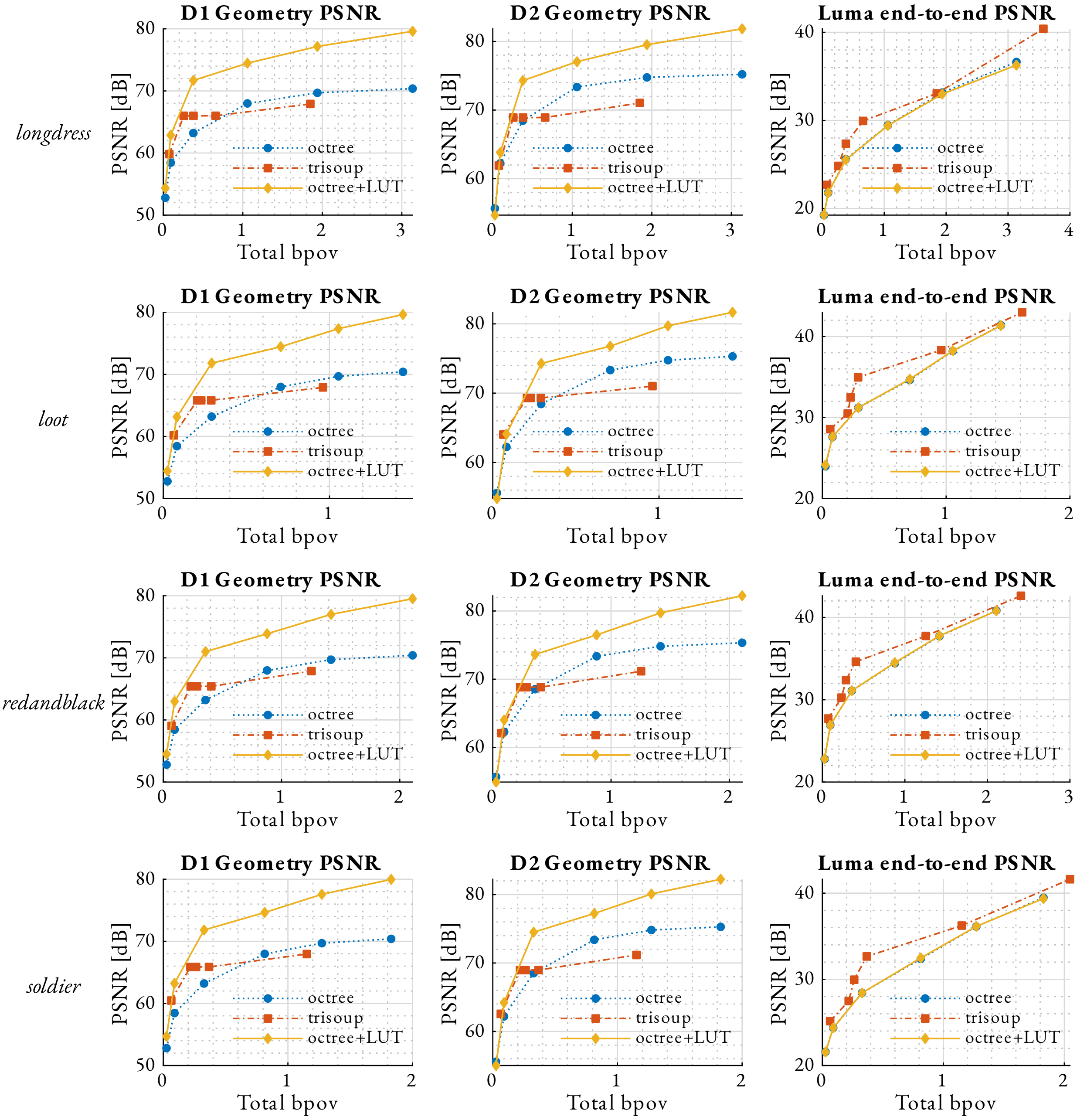


Figure 4.11: Point-based metrics for the interpolative compression application in the *8i_vox10* group.

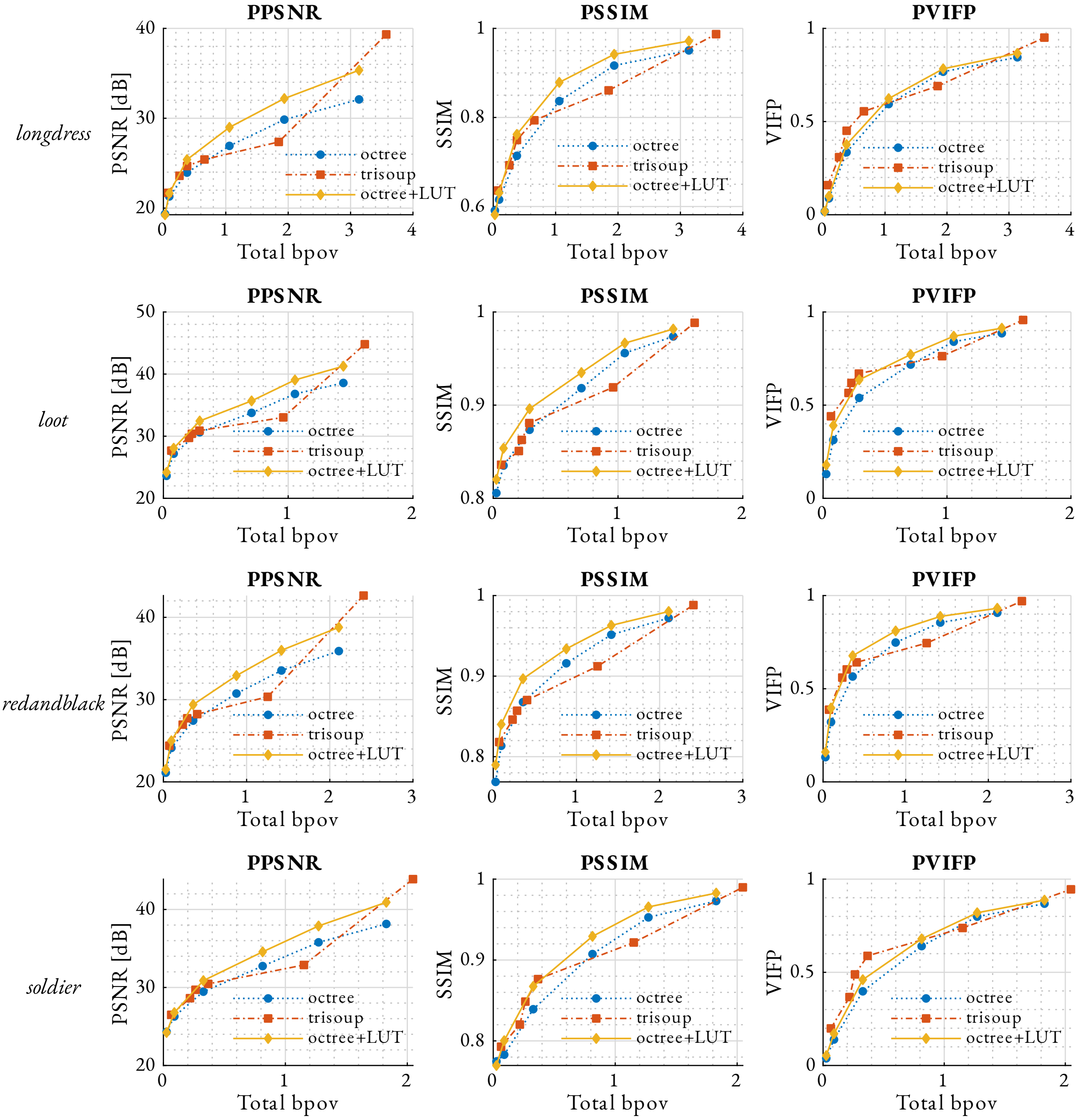
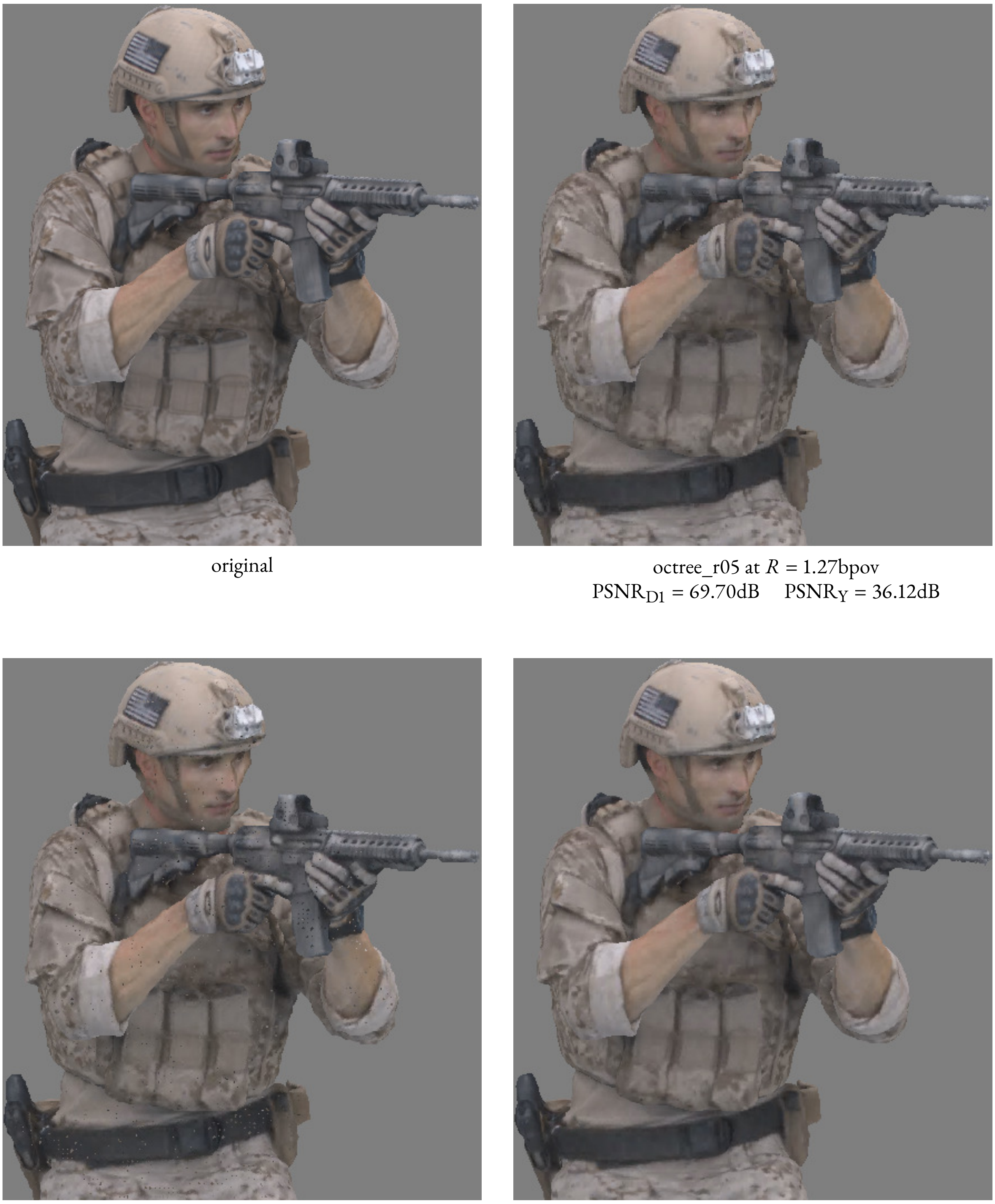


Figure 4.12: Projection-based metrics for the interpolative compression application in the *8i_vox10* group.



original

octree_r05 at R = 1.27bpov
$\mathrm{PSNR}_{\mathrm{D1}}$ = 69.70dB $\mathrm{PSNR}_{\mathrm{Y}}$ = 36.12dB

trisoup_r05 at R = 1.15bpov
$\mathrm{PSNR}_{\mathrm{D1}}$ = 67.95dB $\mathrm{PSNR}_{\mathrm{Y}}$ = 36.24dB

octree_r05+LUT at R = 1.27bpov
$\mathrm{PSNR}_{\mathrm{D1}}$ = 77.57dB $\mathrm{PSNR}_{\mathrm{Y}}$ = 36.14dB

Figure 4.13: Subjective comparison for interpolative compression using *soldier*. We compare geometry and color distortions for the octree, *trisoup*, and the proposed interpolative compression application at around the same bit-rate. The texture was compressed using the RAHT encoder.

5 CONCLUSIONS

A new method was presented for super-resolving intra-frame voxelized point clouds, in which self-similarities of different scales of a point cloud are used to define which of the possible child nodes should be occupied. The SR was performed in fractional downsampled LR point clouds, which can be efficiently represented by the octree structure. Thus, making the proposed method suitable to be used with current state-of-the-art coders.

As no other form of super-resolving intra-frame voxelized point clouds was encountered in the literature, we developed a framework to assess the method's performance. Extensive results show that the proposed method yields lower distortion results when compared to the NNI upsampling and to the NNI upsampling followed by smoothing methods. Furthermore, the method in PCC has shown interesting and competitive results. Although only static point-clouds were used, the proposed method can be easily extended to dynamic ones, with probably even better results [5], as more frames are available to create the LUT.

Moreover, we presented a discussion about techniques to perform fractional resampling of voxelized point clouds, which, although already used in some PCC applications (e.g., TMC13), has not yet been properly presented in the literature.

The bulk of the work in this thesis has been submitted to a journal in July 2020 and is still under revision.

5.1 FUTURE WORK

Future work may focus on robustness against bias in the point cloud geometry and improvements in the super-resolution of sparse point clouds Additionally, a subjective evaluation of the method should be performed. We also plan to improve the super-resolution of texture attributes.

REFERENCES

[1] P. A. Chou. (2018, Jun.) Holograms are the next video. ACM Multimedia Systems Conference. Invited Keynote talk. Accessed: 10-09-2020. [Online]. Available: http://www.mmsys2019.org/downloads/slides/slides-chou.pdf

[2] MPEG, "MPEG Strategic Standardisation Roadmap," ISO/IEC JTC 1/SC 29/WG 11, Geneva, CH, Tech. Rep. N16316, Jun. 2016.

[3] 3DG, "MPEG 3DG and Requirements: Call for proposals for point cloud compression v2," ISO/IEC MPEG JTC1/SC29/WG11, Hobart, AU, Approved WG 11 document N16763, April 2017.

[4] B. Zeng and A. Venetsanopoulos, "A JPEG-based interpolative image coding scheme," in *IEEE International Conference on Acoustics Speech and Signal Processing*. IEEE, 1993.

[5] D. C. Garcia, T. A. Fonseca, R. U. Ferreira, and R. L. de Queiroz, "Geometry coding for dynamic voxelized point clouds using octrees and multiple contexts," *IEEE Transactions on Image Processing*, vol. 29, pp. 313–322, 2019.

[6] E. Alexiou, I. Viola, T. M. Borges, T. A. Fonseca, R. L. de Queiroz, and T. Ebrahimi, "A comprehensive study of the rate-distortion performance in MPEG point cloud compression," *APSIPA Transactions on Signal and Information Processing*, vol. 8, 2019.

[7] A. B. Tucker, Ed., *Computer science handbook*, 2nd ed. Chapman & Hall/CRC, 2004.

[8] A. C. Telea, *Data Visualization*, 2nd ed. Taylor & Francis Ltd., 2014.

[9] M. Levoy. (2014) The Stanford 3D Scanning Repository. Stanford University Computer Graphics Laboratory. Accessed: 15-10-2020. [Online]. Available: http://graphics.stanford.edu/data/3Dscanrep/

[10] G. Turk and M. Levoy, "Zippered polygon meshes from range images," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH '94*. ACM Press, 1994.

[11] C. Tulvan, R. Mekuria, Z. Li, and S. Laserre, "Use Cases for Point Cloud Compression (PCC)," ISO/IEC MPEG JTC 1/SC 29/WG 11, Geneva, CH, Tech. Rep. N16331, Jun. 2016.

[12] S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P. Cesar, P. A. Chou, R. A. Cohen, M. Krivokuća, S. Lasserre, Z. Li, J. Llach, K. Mammou, R. Mekuria, O. Nakagami, E. Siahaan, A. Tabatabai, A. M. Tourapis, and V. Zakharchenko, "Emerging MPEG standards for point cloud compression," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 1, pp. 133–148, 2019.

[13] C. Tulvan and M. Preda, "Point cloud compression for cultural objects," ISO/IEC MPEG JTC1/SC29/WG11, Geneva, CH, Document m37240, Oct. 2015.

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

[15] R. Cohen, H. Ochimizu, D. Tian, and A. Vetro, "Mobile Mapping System Point Cloud Data from Mitsubishi Electric," ISO/IEC MPEG JTC1/SC29/WG11, Hobart, AU, Input Contribution m40495, Apr. 2017.

[16] P. Alliez, F. Forge, L. De Luca, M. Pierrot-Deseilligny, and M. Preda. (2017) Culture 3D Cloud: A Cloud Computing Platform for 3D Scanning, Documentation, Preservation and Dissemination of Cultural Heritage. Accessed: 15-10-2020. [Online]. Available: http://c3dc.fr/

[17] M. Shaw and W. T. and. Scanlab projects. ScanLAB Projects. Accessed: 15-10-2020. [Online]. Available: https://scanlabprojects.co.uk/work/

[18] L. Geosystems. (2019) 3 reasons you should be using point clouds. Leica Geosystems. Accessed: 02-09-2020. [Online]. Available: https://www.youtube.com/watch?v=dlXF3CX0o2Y

[19] FARO. 3d solutions for advanced product design. FARO. Accessed: 02-09-2020. [Online]. Available: https://3ddesign.faro.com/us/product-development/research-development/

[20] A. Collet, M. Chuang, P. Sweeney, D. Gillett, D. Evseev, D. Calabrese, H. Hoppe, A. Kirk, and S. Sullivan, "High-quality streamable free-viewpoint video," *ACM Transactions on Graphics*, vol. 34, no. 4, pp. 1–13, jul 2015.

[21] A. Kipman. (2016) Ted2016: It's a phenomenal time to be human. Microsoft HoloLens. Accessed: 02-09-2020. [Online]. Available: https://blogs.windows.com/devices/2016/03/25/ted2016-its-a-phenomenal-time-to-be-human/

[22] S. Orts-Escolano, C. Rhemann, S. Fanello, W. Chang, A. Kowdle, Y. Degtyarev, D. Kim, P. L. Davidson, S. Khamis, M. Dou, V. Tankovich, C. Loop, Q. Cai, P. A. Chou, S. Mennicken, J. Valentin, V. Pradeep, S. Wang, S. B. Kang, P. Kohli, Y. Lutchyn, C. Keskin, and S. Izadi, "Holoportation: Virtual 3D teleportation in real-time," oct 2016.

[23] 8i. Real human holograms for augmented, virtual and mixed reality. 8i. Accessed: 15-10-2020. [Online]. Available: https://www.8i.com/

[24] R. Mekuria, K. Blom, and P. Cesar, "Design, implementation, and evaluation of a point cloud codec for tele-immersive video," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 4, pp. 828–842, apr 2017.

[25] R. Pagés, K. Amplianitis, D. Monaghan, J. Ondřej, and A. Smolić, "Affordable content creation for free-viewpoint video and VR/AR applications," *Journal of Visual Communication and Image Representation*, vol. 53, pp. 192–201, may 2018.

[26] Intel. Intel true view. Intel Corporation. Accessed: 15-09-2020. [Online]. Available: https://www.intel.com.br/content/www/br/pt/sports/technology/true-view.html

[27] Mitsubishi. Mobile Mapping Systems (MMS: High precision gps mobile measuring equipament. Mitsubishi. Accessed: 15-10-2020. [Online]. Available: http://www.mitsubishielectric.com/bu/mms/

[28] D. Flynn, S. Lasserre, and G. Martin-Cocher, "PCC Cat3 test sequences from BlackBerry|QNX," ISO/IEC MPEG JTC1/SC29/WG11, Ljubljana, Slovenia, Input Document m23647, Jul. 2018.

[29] S. Chen, B. Liu, C. Feng, C. Vallespi-Gonzalez, and C. Wellington, "3d point cloud processing and learning for autonomous driving."

[30] M. Levoy. (1999) The Digital Michelangelo Project. Stanford University. Accessed: 15-10-2020. [Online]. Available: https://accademia.stanford.edu/mich/

[31] 8i. (2017, Sep.) Holo with ARKit is here, bringing volumetric human holograms to the masses on the iPhone. 8i. Accessed: 15-10-2020. [Online]. Available: https://link.medium.com/fpSGbpvjDab

[32] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, no. 1/3, pp. 7–42, 2002.

[33] H. Kim, I. Kitahara, K. Kogure, and K. Sohn, "A real-time 3d modeling system using multiple stereo cameras for free-viewpoint video generation," in *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006, pp. 237–249.

[34] C. Loop, C. Zhang, and Z. Zhang, "Real-time high-resolution sparse voxelization with application to image-based modeling," in *Proceedings of the 5th High-Performance Graphics Conference on - HPG '13*. ACM Press, 2013.

[35] M. Gross and H. Pfister, Eds., *Point-Based Graphics*. Morgan Kaufmann Publishers, 2007.

[36] A. Parker, "Stereoscopic vision," in *Encyclopedia of Neuroscience*. Elsevier, 2009, pp. 411–417.

[37] Wikipedia contributors. (2020, Sep.) Depth perception. Online. Wikipedia, The Free Encyclopedia. Accessed: 11-09-2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Depth_perception&oldid=976150693

[38] R. S. Laramee. (2019) Data visualization classes 2019. Computer Science Department at Swansea University. Accessed on: 04-09-2020. [Online]. Available: http://www.youtube.com/user/rslaramee/

[39] V. Castelli and L. D. Bergman, Eds., *Image Databases*. John Wiley & Sons, 2001.

[40] R. Szeliski, *Computer Vision*. Springer London, 2011.

[41] R. Gonzalez, *Digital image processing*, 3rd ed. Upper Saddle River, N.J: Prentice Hall, 2008.

[42] ITU-R BT.709-6, "Parameter values for the hdtv standards for production and international programme exchange," International Telecommunication Union, Recommendation, May 2015.

[43] M. Gross, "Getting to the Point...?" *IEEE Computer Graphics and Applications*, vol. 26, no. 5, pp. 96–99, sep 2006.

[44] M. Zwicker, H. Pfister, J. van Baar, and M. Gross, "Surface splatting," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH '01*, ser. SIGGRAPH '01. New York, NY, USA: ACM Press, 2001, p. 371–378.

[45] M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt, "High-quality surface splatting on today's GPUs," in *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005.* IEEE, 2005, pp. 17–141.

[46] Wikibooks contributors. (2020, Apr.) Cg Programming/Unity/Billboards. Online. Wikibooks, The Free Textbook Project. Accessed: 28-12-2020. [Online]. Available: https://en.wikibooks.org/w/index.php?title=Cg_Programming/Unity/Billboards&oldid=3678770

[47] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "MeshLab: an Open-Source Mesh Processing Tool," in *Eurographics Italian Chapter Conference*, V. Scarano, R. D. Chiara, and U. Erra, Eds. The Eurographics Association, 2008.

[48] D. Girardeau. Cloudcompare: 3D point cloud and mesh processing software. open source project. Accessed: 14-09-2020. [Online]. Available: https://www.cloudcompare.org/

[49] MATLAB. pcshow: Plot 3-d point cloud. MathWorkds. Accessed: 15-09-2020. [Online]. Available: https://www.mathworks.com/help/vision/ref/pcshow.html

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

[51] C. Guede, J. Ricard, S. Lasserre, and J. Llach, "Technicolor point cloud renderer," ISO/IEC MPEG JTC 1/SC 29/WG 11, Paris, doc. m40229, July 2017.

[52] Nurulize. (2017) Atom view. Sony. Accessed: 15-09-2020. [Online]. Available: https://vimeo.com/user59248173

[53] Atomontage. (2018) Atomontage media. Atomontage. Accessed: 15-09-2020. [Online]. Available: https://www.atomontage.com/media

[54] M. Levoy and T. Whitted, "The use of points as a display primitive," Computer Science Department, The University of North Carolina at Chapel Hill, Tech. Rep. 85-022, Jan. 1985.

[55] Wikipedia contributors. (2020, Sep.) Kinect. Online. Wikipedia, The Free Encyclopedia. Accessed: 22-09-2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Kinect&oldid=979120984

[56] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (PCL)," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, may 2011.

[57] R. Mekuria, M. Sanna, S. Asioli, E. Izquierdo, D. C. A. Bulterman, and P. Cesar, "A 3D tele-immersion system based on live captured mesh geometry," in *Proceedings of the 4th ACM Multimedia Systems Conference on - MMSys '13*. ACM Press, 2013.

[58] P. Fechteler, R. Mekuria, P. Cesar, D. Monaghan, N. E. O'Connor, P. Daras, D. Alexiadis, T. Zahariadis, A. Hilsmann, P. Eisert, S. V. Broeck, C. Stevens, J. Wall, M. Sanna, D. A. Mauro, and F. Kuijk, "A framework for realistic 3d tele-immersion," in *Proceedings of the 6th International Conference on Computer Vision / Computer Graphics Collaboration Techniques and Applications - MIRAGE '13*. ACM Press, 2013.

[59] R. Mekuria, C. Tulvan, and Z. Li, "Requirements for point cloud compression," ISO/IEC MPEG JTC1/SC29/WG11, Geneva, CH, MPEG Requierements w16330, Feb. 2016.

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

[61] 3DG, "V-PCC Codec Description," ISO/IEC MPEG JTC1/SC29/WG11, Geneva, CH, Approved WG 11 document N18892, Oct. 2019.

[62] ——, "G-PCC codec description v5," ISO/IEC MPEG JTC1/SC29/WG11, Geneva, CH, Approved WG 11 document N18891, Oct. 2019.

[63] G. Sullivan and J.-R. Ohm, "Meeting report of the 13th meeting of the joint collaborative team on video coding (jct-vc)," ITU-T SG16 WP3 and ISO/IEC MPEG JTC1/SC29/WG11, Incheon, KR, Report JCTVC-M1000, Apr. 2013.

[64] Wikipedia contributors. (2020) Versatile Video Coding. Wikipedia, The Free Encyclopedia. Accessed: 22-09-2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Versatile_Video_Coding&oldid=979632884

[65] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," *SIGGRAPH Comput. Graph.*, vol. 26, no. 2, p. 71–78, Jul. 1992.

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

[67] 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, aug 2016.

[68] R. L. de Queiroz, D. C. Garcia, P. A. Chou, and D. A. Florencio, “Distance-based probability model for octree coding,” *IEEE Signal Processing Letters*, vol. 25, 2018.

[69] S. Lasserre and D. Flynn, “[PCC] Inference of a mode using point location direct coding in TMC3,” ISO/IEC MPEG JTC1/SC29/WG11, Gwangju, Korea, Input contribution m42239, Jan. 2018.

[70] P. A. Chou, “Trisoup C++ reference code for TMC13,” ISO/IEC MPEG JTC1/SC29/WG11, Ljubjana, Slovenia, Input Document m43786, Jul. 2018.

[71] 3DG, “PCC Test Model Category 3 v0,” ISO/IEC MPEG JTC1/SC29/WG11, Macau, China, Approved WG 11 document w17249, Oct. 2017.

[72] K. Mammou, A. Tourapis, J. Kim, F. Robinet, V. Valentin, and Y. Su, “Lifting Scheme for Lossy Attribute Encoding in TMC1,” ISO/IEC MPEG JTC1/SC29/WG11, San Diego, US, Input contribution m42640, Apr. 2018.

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

[74] G. Sandri, R. L. de Queiroz, and P. A. Chou, “Comments on "Compression of 3D Point Clouds Using a Region-Adaptive Hierarchical Transform".”

[75] S. Lasserre and D. Flynn, “[G-PCC][new proposal] On an improvement of RAHT to exploit attribute correlation,” ISO/IEC MPEG JTC1/SC29/WG11, Geneva, CH, Tech. Rep. m47378, Mar. 2019.

[76] ——, “G-PCC CE13.18 report on upsampled transform domain prediction in RAHT,” ISO/IEC MPEG JTC1/SC29/WG11, Gothenburg, Sweden, Input document m49380, Jul. 2019.

[77] W. Sweldens, “The Lifting Scheme: A Custom-Design Construction of Biorthogonal Wavelets,” *Applied and Computational Harmonic Analysis*, vol. 3, no. 2, pp. 186–200, apr 1996.

[78] A. Javaheri, C. Brites, F. Pereira, and J. Ascenso, “Point cloud rendering after coding: Impacts on subjective and objective quality.”

[79] D. Tian, H. Ochimizu, C. Feng, R. Cohen, and A. Vetro, "Updates and Integration of Evaluation Metric Software for PCC," ISO/IEC MPEG JTC1/SC29/WG11, Hobart, AU, Input Document m40522, Apr. 2017, 2017.

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

[81] Wikipedia contributors. (2020, Oct.) Hausdorff distance. Wikipedia, The Free Encyclopedia. Accessed: 01-10-2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Hausdorff_distance&oldid=971025942

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

[83] D. Tian, H. Ochimizu, C. Feng, R. Cohen, and A. Vetro, "Evaluation Metrics for Point Cloud Compression," ISO/IEC MPEG JTC1/SC29/WG11, Chengdu, China, Input Document m39316, Oct. 2016.

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

[85] 3DG, "Common test conditions for point cloud compression," ISO/IEC MPEG JTC1/SC29/WG11, Gothenburg, SE, Approved WG 11 document N18883, July 2019.

[86] J.-R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, "Comparison of the Coding Efficiency of Video Coding Standards—Including High Efficiency Video Coding (HEVC)," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1669–1684, dec 2012.

[87] R. L. de Queiroz and P. A. Chou, "Motion-compensated compression of dynamic voxelized point clouds," *IEEE Transactions on Image Processing*, vol. 26, no. 8, pp. 3886–3895, Aug 2017.

[88] E. Alexiou and T. Ebrahimi, "Exploiting user interactivity in quality assessment of point cloud imaging," in *2019 Eleventh International Conference on Quality of Multimedia Experience (QoMEX)*. IEEE, jun 2019.

[89] T.-Y. Kuo, Y.-J. Wei, and K.-H. Wan, "Color Image Quality Assessment Based on VIF," in *2019 3rd International Conference on Imaging, Signal Processing and Communication (ICISPC)*. IEEE, jul 2019.

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

[91] LIVE – Laboratory for Image & Video Engineering. Image & Video Quality Assessment Algorithms. The University of Texas at Austin. Accessed on: 28-10-2020. [Online]. Available: https://live.ece.utexas.edu/research/Quality/index_algorithms.htm

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

[93] H. R. Sheikh and A. C. Bovik. Image Information and Visual Quality. The University of Texas at Austin. Accessed: 28-10-2020. [Online]. Available: https://live.ece.utexas.edu/research/Quality/VIF.htm

[94] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*. ADDISON WESLEY PUB CO INC, 1994. [Online]. Available: https://www.ebook.de/de/product/3236755/ronald_l_graham_donald_e_knuth_oren_patashnik_concrete_mathematics_a_foundation_for_computer_science.html

[95] R. Bridson, "Fast Poisson disk sampling in arbitrary dimensions," in *ACM SIGGRAPH 2007 sketches on - SIGGRAPH '07*. ACM Press, 2007.

[96] A. Nealen. (2004, May) An as-short-as-possible introduction to the least squares, weighted least squares and moving least squares methods for scattered data approximation and interpolation. [Online]. Available: http://www.nealen.de/projects/mls/asapmls.pdf

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

[98] A. C. Öztireli, G. Guennebaud, and M. Gross, "Feature preserving point set surfaces based on non-linear kernel regression," *Computer Graphics Forum*, vol. 28, no. 2, pp. 493–501, apr 2009.

[99] H. Huang, S. Wu, M. Gong, D. Cohen-Or, U. Ascher, and H. R. Zhang, "Edge-aware point set resampling," *ACM Trans. Graph.*, vol. 32, no. 1, Feb. 2013.

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

[101] C. Dinesh, G. Cheung, and I. V. Bajić, "Super-resolution of 3D color point clouds via fast graph total variation," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 1983–1987.

[102] A. Hamdi-Cherif, J. Digne, and R. Chaine, "Super-resolution of point set surfaces using local similarities," *Computer Graphics Forum*, vol. 37, no. 1, pp. 60–70, jun 2017.

[103] D. C. Garcia, T. A. Fonseca, and R. L. de Queiroz, "Example-based super-resolution for point-cloud video," in *IEEE International Conference on Image Processing (ICIP)*. IEEE, 2018, pp. 2959–2963.

[104] A. B. Yutaka and E. B. Y. Ohtake, "A comparison of mesh smoothing methods," in *In Proceedings of the Israel-Korea BiNational Conference on Geometric Modeling and Computer Graphics*, 2003, pp. 83–87.

[105] R. Cabello. (2020) Three.js – JavaScript 3D library. Accessed: 07-12-2020. [Online]. Available: https://threejs.org/

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

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

[108] P. A. Chou and M. Krivokuca, "Concerns about objective metrics for point cloud compression," ISO/IEC MPEG JTC 1/SC 29/WG 11, Macau, Input document to AhG on Point Cloud Compression m41809, October 2017.