DOI: 10.1111/cgf.15130

Eurographics Symposium on Geometry Processing 2024
R. Hu and S. Lefebvre
(Guest Editors)

COMPUTER GRAPHICS *forum*
*Volume 43 (2024), Number 5*

# Entropy-driven Progressive Compression of 3D Point Clouds

A. Zampieri[1,2], G. Delarue[2], N. Abou Bakr[2], and P. Alliez[1]

[1] Inria Sophia antipolis, France
[2] Samp.ai Paris, France

**Abstract**

*3D point clouds stand as one of the prevalent representations for 3D data, offering the advantage of closely aligning with sensing technologies and providing an unbiased representation of a measured physical scene. Progressive compression is required for real-world applications operating on networked infrastructures with restricted or variable bandwidth. We contribute a novel approach that leverages a recursive binary space partition, where the partitioning planes are not necessarily axis-aligned and optimized via an entropy criterion. The planes are encoded via a novel adaptive quantization method combined with prediction. The input 3D point cloud is encoded as an interlaced stream of partitioning planes and number of points in the cells of the partition. Compared to previous work, the added value is an improved rate-distortion performance, especially for very low bitrates. The latter are critical for interactive navigation of large 3D point clouds on heterogeneous networked infrastructures.*

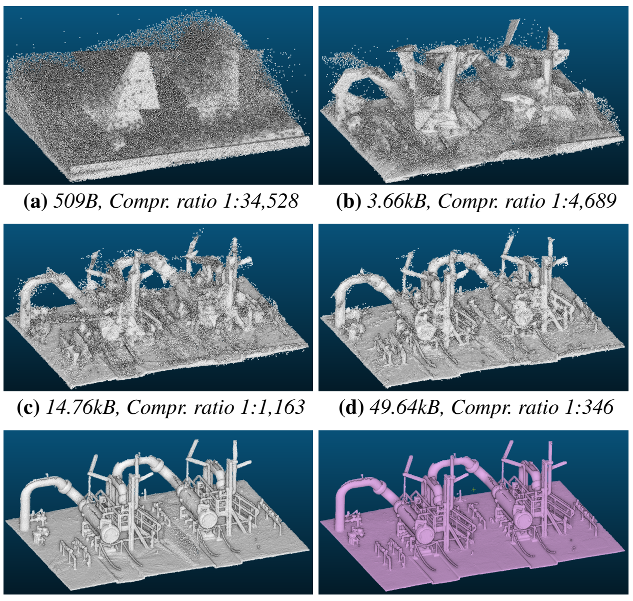
**CCS Concepts**

***• Mathematics of computing → Coding theory; • Applied computing → Computer-aided design; • Theory of computation → Data compression; • Computing methodologies → Point-based models;***

## 1. Introduction

Point clouds, as 3D representations of objects or scenes, have become ubiquitous in various fields such as reverse engineering, remote sensing or digital twinning. The accessibility of high-resolution 3D sensors and scanning technologies has led to a fast growth in the size of 3D point clouds, posing significant challenges for storage and transmission. Large point clouds also hamper real-time remote applications such as visualization and navigation, due to the latency induced by streaming large volumes of data. In response to these challenges, progressive compression techniques have become crucial for dealing with large 3D point clouds. The progressive compression of 3D point clouds involves reducing the amount of data required to represent a given scene, while preserving its geometric information and providing intermediate representations of the input data. Intermediate herein often translates into incomplete (less points), or inaccurate (points with reduced accuracy - see Figure 1), or even alternate representations. Effective progressive compression techniques not only minimize storage and bandwidth requirements, but also facilitate real-time partial rendering and processing. A common objective is to convert the input 3D point cloud into a stream of refinements, while optimizing a curve plotting the rate-distortion trade-off.

In this work, we tackle progressive compression of raw, static 3D point clouds, with a focus on scalability for low bit-rates. More specifically, we seek for satisfactory rate-distortion curves at the beginning of the transmission. This is an enduring problem because prediction is harder at low bitrates, and the high variability of 3D point clouds corresponds to high data entropy. For low bitrates, a data modeling approach is crucial to highly compress the input 3D point cloud. We review next the related work focused on single-rate or progressive compression of 3D point clouds.



**(a)** *509B, Compr. ratio 1:34,528* **(b)** *3.66kB, Compr. ratio 1:4,689* **(c)** *14.76kB, Compr. ratio 1:1,163* **(d)** *49.64kB, Compr. ratio 1:346* **(e)** *247.69kB, Compr. ratio 1:69* **(f)** *17,163kB, No compression*

**Figure 1:** *Progressive decompression of a 3D point cloud acquired on an industrial site (2M points, depicted in pink). The point cloud is progressively reconstructed by sampling cells of a general (refining) binary space partition. Our approach improves rate-distortion tradeoffs even on very low bit-rates.*

© 2024 The Authors. Computer Graphics Forum published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.
This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

## 1.1. Related Work

Since the advent of point-based graphics, 3D point clouds are increasingly popular. Static or dynamic point clouds are used in a wide range of industrial applications such as digital twinning or telepresence.

The compression of 3D point clouds has gained some attention especially in scenarios where efficient storage and transmission is required. Multiple variants of compression types and objectives exist, we refer to a comprehensive survey on this topic [CPZ19]. In our taxonomy, we first distinguish between lossy and lossless compression - the former offering in general higher compression ratios at the cost of decreased accuracy. Low processing cost facilitates deployment over limited or heterogeneous architectures. Progressiveness is mandatory for scalable, real-time applications on networked infrastructures . Compression is also applied to point clouds with attributes such as colors or semantic labels.

Good compression ratios are often achieved by leveraging concepts from information theory and geometry, taking advantage of specific properties of 3D point clouds. We explore next some of those concepts.

**Entropy coding.** Entropy coding is a widely adopted method for generic data compression [Huf52]. Based on information theory, the entropy of a source of symbols is a measure of the uncertainty or randomness inherent in the input data. It quantifies the average amount of information produced by the source, per symbol. It also provides an upper bound on the number of bits required to represent each symbol of the source efficiently. An entropic coder leverages the distribution of symbols to approach the optimal compression ratios by assigning short codes to frequent symbols, and vice-versa. An *adaptive* entropic coder updates the distribution of symbols during encoding, to avoid transmitting a priori the full distribution.

However, the distribution of symbols is less intuitive to define for 3D point clouds. Encoding the floating point coordinates is possible [ILS05], but the distribution of point coordinates is sparse and is unlikely to exhibit significant repetitions that can benefit entropic coding. Vector quantization [Gra84] is a common transform that generates a spatially coherent distribution of symbols. However, it still carries an additional cost resulting from the sequence order, proportional to the number of possible permutations ($log_2(n!)$ bits for $n$ non-duplicated points.) This information is useless and shall be discarded by the encoder. Finding an adequate transform is an open problem and various solutions have been proposed.

**1D ordering.** A first line of approaches pioneered by Gumhold et al. [GKIS05] compute a spatially coherent order of the points via spanning trees. Such trees facilitate prediction of future points from previously encoded ones. Additionally, the residuals (corrective vectors after prediction) exhibit a low excursion and highly unbalanced distribution, making entropy encoding effective. This suggests a predictive delta encoding where the difference between the prediction and the next symbol is encoded. A predictor paired with a tailored quantization model tends to skew the distribution of these delta-coded symbols towards zero, with lower entropy. While simple and effective, this approach does not offer progressiveness.

**2D Projection.** Another line of approaches consists in using 2D projections of the input points onto a set of planar surfaces. Such projections can leverage well-known image or video encoders to compress depth maps with attributes that can be reconstructed into point clouds. These approaches were introduced for compression [OS04] as well as for spectral analysis [PG01]. Spectral representations are leveraged for efficient encoding of depth information and for dynamic point clouds by taking advantage of spatio-temporal coherency [MSMW07]. The latter approach has been adopted by the Moving Picture Expert Group (MPEG) as one of the standards for point cloud compression [SPB*18].

**3D partitioning.** Another popular representation relies on iterative space partitioning. Octrees are quite popular among those representations. In this model, the bounding box of the input point cloud is recursively subdivided into eight regular spatial regions (octants), and only the (binary) occupancy of each octant is encoded. The recursive nature of octree offers progressiveness. Such a data model was utilized for compression [PK03] as it offers several relevant properties: (1) Each empty region does not require further bisection, preventing the waste of occupancy bits there (see empty regions in 2a); (2) Occupancy bits of an octree factors out the encoding of all points of an octant. More specifically, each octant encodes the most significant bit (on each dimension) of all its enclosed points. The first level of octree decomposition thus encodes using 8 bits an equivalent of $3n$ bits where $n$ denotes the number of input points . Octrees are thus similar in spirit to the Haar wavelet transform [Haa09] proposed for image compression [Mal89], but deal with points coordinates instead of pixel intensities.

Octree-based methods are especially popular for compression of static point clouds [SK06, ZGL20, QLX21], and several variants have been proposed. Some approaches apply predictive encoding using the already decoded context and local geometric properties such as continuity or planarity [SK06]. Other approaches utilize the occupancy information of neighboring octants as prior to predict the occupancy probability of the current octant [QLX21]. Such an occupancy predictor is combined with another predictor for planar sections to guide the probability model. Octant traversal is also ordered so as to favor occupancy prediction [HPKG06, JTC*12]. Some approaches combine quadtrees with binary trees [KLL*18], or simpler binary space partitions [DG00] to increase factorization and improve the final rate-distortion tradeoffs.

Octrees offer an advantage over generic BSP data structures by implicitly defining bisecting primitives. They, However, cannot adapt to the object geometry. A generic BSP data structure must encode the bisecting primitive.It can however adapt to the geometrical properties of the object. One approach consist in iteratively partitioning a point cloud based on its local PCA properties [BLL*06]. The bisecting plane is encoded with Euler angle and its distance to origin. Although this method requires more bits for plane encoding, the cost is compensated by the adaptation to the point cloud geometry. Others used a more rigid method where the direction remain implicitly defined but the plane position can vary [ZXLL17].

**Standardization.** The popularity of 3D point clouds for real-world applications motivated the emergence of two MPEG standards [ISO12]. They correspond to variations of the aforementioned solutions: an octree-based encoder referred to as G-PCC suited to sparse point distributions, and a planar-projection-based approach referred to as V-PCC [JPM*19] suited to uniform dense dynamic 3D point clouds. The Joint Picture Expert group (JPEG) also records a standardization proposal based on voxelization and encoding via deep learning [GRR*22]. Finally, the interest of industrial companies for 3D point clouds is confirmed by patents on compression methods [MTS*22, WLHS23, JDL*21].

**Deep learning** Deep learning approaches also made their way into point cloud compression. Autoencoders are relevant for lossy encoding [QVD19], where residual vectors after prediction are quantized and encoded using neural networks. Neural networks have been used to predict the octant occupancy in octree representations [NQVD21], or to predict the probability distribution of octants [QLX21]. These approaches are however complex to train as each new sensing modality often requires retraining of the network. They are also more memory- and compute-intensive than previous approaches in general.

**LiDAR.** Some approaches are tailored to LiDAR point clouds [TTMT16], based on spectral analysis [PG01]. The main idea is to take advantage of the regularity (in spherical coordinates) of the LiDAR sampling process, and of prior knowledge on the sensing sequence to recast the problem as the one of compression of depth images. Other approaches leverage different image encoding methods such as autoencoders [TTCT19].

**Attributes.** A related topic is the compression of point cloud attributes such as colors, reflectivity or semantic labels. Albeit this topic is outside the scope of this paper, we mention two approaches based on graph transforms [ZFL14] and Haar wavelets [DQC16].

**Discussion.** As discussed above, many approaches rely on spatial subdivisions and related transforms. As most of these subdivisions are both axis-aligned and regular (bisecting cells symmetrically), they are suboptimal for most real-world cases. Consider e.g., such a 3D space partition [GD02]. The related compression approach is effective when many points are located in one cell, the optimal case being when all points are superimposed - a degenerate case, and the worst case being a uniform point cloud. A simple point cloud sampled on a horizontal 2D segment already requires many subdivision levels, see Figure 2(a). The localization capability is even lower when the sampled 2D segment is not axis-aligned - see Figure 2(b), due to aliasing. Using non-axis-aligned, non-regular subdivision is more effective (Figure 2(c)).

Another degree of freedom lies into the ordering of the bisections. The aforementioned approach [GD02] relies on breadth first canonical ordering (see Figure 2a). The virtue of such an ordering is to provide a decreasing error bound during decoding. However, it does not prioritize high density cells containing most of the information. This has a negative impact on rate-distortion since the encoder spends bits for almost empty cells.

### 1.2. Positioning and contributions

Departing from standard axis-aligned data structures, we design an approach that solves the aforementioned issues. Our framework is

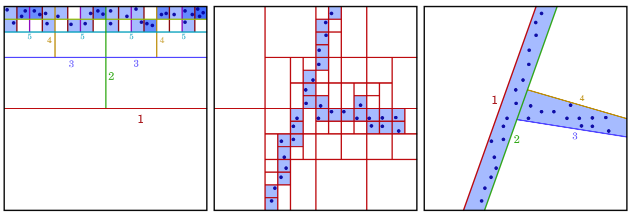


**Figure 2:** *Binary space partitioning in 2D. (a) Regular, axis-aligned space bisection (favorable case), approximation with 5 levels of bisection (10 bisecting planes). (b) Axis-aligned regular space bisection (unfavorable case), approximation with 8 levels of bisection (72 bisecting planes). (c) Non axis-aligned, non-regular space bisection, approximation with 4 levels of bisection (4 bisecting planes).*

inspired by the elegant "cardinality" approach pioneered by Devillers and Gandoin [DG00], which performs a recursive space subdivision while encoding the number of points in each cell. In their approach, space subdivision is a kD-tree induced by symmetric, axis-aligned planes, and canonical ordering of the bisections. This approach yields satisfactory rate-distortion ratios for a low computational complexity, and was implemented in the Draco geometry compression library [Dra]. Our approach utilizes arbitrary planes instead, which are optimized from an entropy criterion. In addition, we devise an adaptive ordering of the bisections. Relying upon arbitrary planes offers novel degrees of freedom but comes at a cost, as we must encode them. We devise a novel encoding scheme for 3D planes that combines adaptive quantization and prediction. Our experiments demonstrate that such an approach yields significant improvements in terms of compression ratios and rate-distortion trade-offs.

In summary, our contributions are:

- **Algorithm:** a novel way to encode an input unstructured 3D point cloud based on general binary space partitioning.
- **Optimization:** a novel entropy-based metric that governs the choice of bisecting 3D planes.
- **Encoding:** an adaptive quantization and prediction approach based on Fibonacci curves and principal covariance analysis of 3D cells, tailored to encode 3D planes.

## 2. Our Approach

We now define several terms:

- Cell: Refers to a convex 3D polyhedral cell of the binary space partition.
- Bisection: Action of subdividing a cell into two sub-cells.
- Cardinality: Number of points in a cell.
- BSP: Binary space partition.

### 2.1. Overview

Our approach takes a 3D point cloud as input, and generates as output a stream of refinements. For encoding, we start by computing a root cell - by default the bounding box of the input points - and optimize a binary space partition of it by recursive bisection. The output stream is constructed by encoding and interlacing the bisecting planes and cardinality information of each BSP bisection. Bisecting planes are chosen based on an entropy criterion, and adaptively quantized and predicted based on the shape of the current bisected cell. During decoding, the binary space partition is iteratively refined, and we uniformly sample points in each cell (where the number of sampled points for a cell equates its decoded cardinality.) The distortion of such a progressive compression is measured between the input point cloud and a sampling of the cells of the BSP based on the cardinality information. Figure 3 depicts an overview of our approach in 2D.

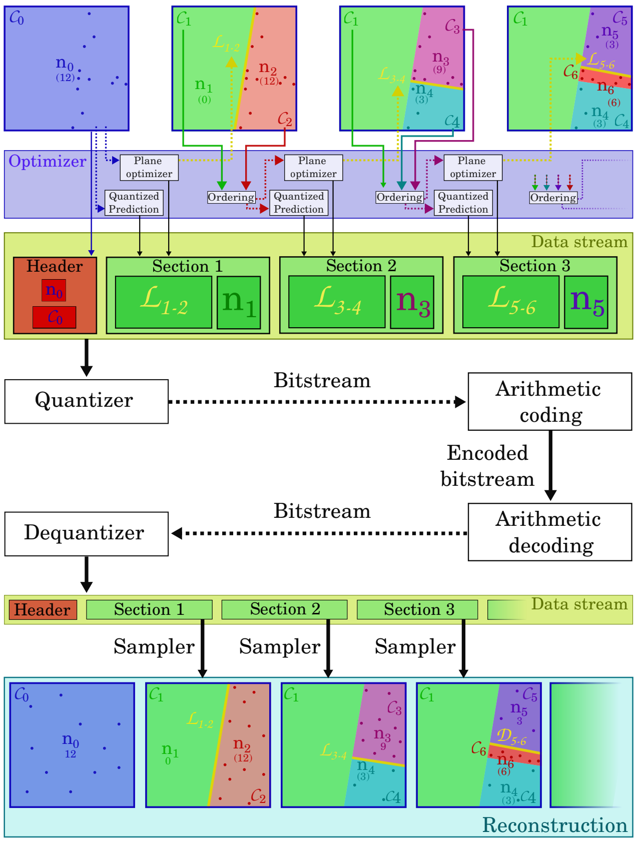


**Figure 3:** *Overview of our approach in 2D. Top: Iterative bisection of cells organized in a specific order. The data of each bisection (planes and cardinality) are organized into a stream (middle-top) that is quantized and encoded (middle). Bottom: The decoder recovers the encoded bisecting lines and cardinality information, and refines the binary space partition. The point cloud is reconstructed by uniform sampling of the cells.*

The atomic operator of our approach bisects a cell into two subcells using a bisecting plane. It then encodes the bisecting plane and the cardinality in one of the two sub-cells, the other cardinality being implicitly deduced. The bisection process recurs until either the cell cardinality is smaller than a user-defined threshold (set by default to 3 points), or a maximum user-defined tolerance error is met (by default set to the sampling radius distance, defined later). We consider this tolerance to be met when the diameter of the cell (maximal distance between cell vertices) is smaller than the tolerance. Two examples of such atomic bisections are depicted by Figure 5.

**Encoder.** The complete encoding algorithm proceeds offline as follows: The sequence of optimal (quantized) bisecting planes is generated in tandem with the construction of the binary space partition. We obtain two sequences: the quantized bisecting planes and residuals of predicted cardinality. These two sequences are then interlaced and encoded using arithmetic encoding [WNC87].

**Decoder.** The complete decoding algorithm proceeds online as follows. It first reads the header containing the bounding box and total cardinality. It then decodes the bitstream to recover the bisecting planes and cardinality values to reconstruct the BSP-tree. At any time during decoding, the cells can be uniformly sampled to reconstruct a 3D point cloud.

## 2.2. Bisecting Planes

The elegant idea behind the "cardinality" transform [DG00] is that encoding the number of points inside a cell (of a progressively refined partition) instead of the point coordinates provides information for all points in the said cell. Intuitively, it compresses by factoring out more bits of accuracy for all points in the cell. Such a factorization principle is even more effective when the input point cloud is non-uniform. We leverage such a non-uniformity principle to determine the optimal bisecting plane by maximizing the difference of density between sub-cells after bisection. Intuitively, the optimal planes must tightly "sandwich" points sampled on a planar surface part, or loosely sandwich points sampled on a curved surface part. This is feasible by adequately choosing the bisecting plane. The plane selection must allow the folowing operation: Carving large empty space if any, splitting two sampled parts separated by empty space, or splitting dense parts. In our initial experiments, we realized that the repertoire of such required operators is potentially infinite. The definition of an optimal bisecting plane being an ill-posed problem, we analyze instead the information contained in a 3D point cloud based on the notion of entropy.

### 2.2.1. Entropy-based metric

A first step consists in quantifying the information transmitted per point. In the naive approach where points are encoded sequentially, this quantity is evaluated by counting the number of bits transmitted, divided by the number of points. In the case of a regular, axis-aligned binary space partition, a cell is symmetrically subdivided into two subcells of equal volume, and the cardinality of one subcell is encoded [DG00]. An analysis shows that the number of bits per point encoded at each bisection level is 1. Each bisection separating a cell along its most important dimension into two sub-cells, this is equivalent to providing the next most significant bit of a uniform quantization scheme for all points. We now wish to address the analog for general binary space partitions.

We first propose $L_b$, to calculate the total number of bits encoded:

$$L_b = n_1 \cdot \log_2\left(\frac{V_t}{V_1}\right) + n_2 \cdot \log_2\left(\frac{V_t}{V_2}\right), \tag{1}$$

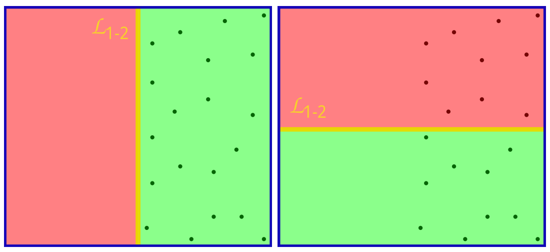
where $n_i$ denotes the number of points in partition $i$, $V_i$ denotes the volume of partition $i$ and $V_t$ denotes the total volume $V_1 + V_2$.

This quantity is derived from the fact that one symmetrical axis aligned subdivision encode for each point one bit of information $\left(\log_2\left(\frac{V_t}{1/2 \cdot V_T}\right)\right)$ and from the definition of the Shannon entropy from information theory [Sha48]:

$$E_{\mathcal{S}} = p(X) \cdot \log_2(p(X)), \tag{2}$$

where $p(X)$ denotes the probability of symbol $X$.

We can verify that Equation 1 is consistent with the aforementioned quantity for a symmetric bisection where $V_1 = V_2 = V_t/2$. This equation results from transforming the information of per-point localization into a number of bits. However, this equation is insufficient for driving a bisection scheme. For example, computing Equation 1 as shown by Figure 4 would yield the same entropy for both sub-cells after bisection, while the bisection shown Figure 4(a) is more effective at localizing points.



**(a)** **(b)**

**Figure 4:** *Naive entropy-driven bisection. Both bisections (a) and (b) correspond to the same entropy while (a) better localizes the points by generating one empty and one dense sub-cell.*

The issue comes from defining a pure per-point metric: Equation 1 represents for each point the amount of localization information provided independently from their spatial distribution. An ideal bisection would however optimize the localization of the whole distribution.

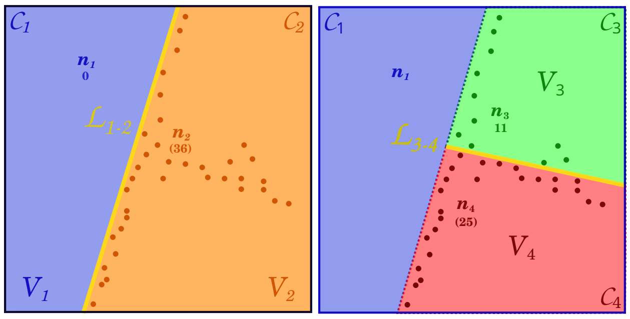


**Figure 5:** *Two bisection steps. We initially bisect the bounding box ($\mathcal{C}_1 \cup \mathcal{C}_2$) into $\mathcal{C}_1$ and $\mathcal{C}_2$, then bisect $\mathcal{C}_2$ into $\mathcal{C}_3$ and $\mathcal{C}_4$ (note that bisecting $\mathcal{C}_1$ is not necessary since $n_1 = 0$). $n_k$ denotes the number of points in $\mathcal{C}_k$, $V_k$ denotes the volume of $\mathcal{C}_k$ and $\mathcal{L}_{(k)-(l)}$ denotes the bisection plane between $\mathcal{C}_k$ and $\mathcal{C}_l$.*

We now analyze the notion of point cloud information. While the localization of 3D points in space is informative, the localization of empty space is equally informative. The following entropy metric views a point cloud as a density distribution $p \in \mathbb{R}^3$. The entropy of such a distribution can be defined as the differential entropy [Sha48]:

$$E_0(p) = \int_{X \in V} p(X) \cdot \log_2(p(X))dX, \tag{3}$$

where $p$ denotes the probability density distribution, and $V$ denotes the probability domain.

From a given density distribution function, considering an estimation of this distribution $\hat{p}$, we can define a cross entropy function. This provides a measure of the entropic distance (encoding cost) between the estimated distribution and the real one. The cross entropy is estimated as:

$$E_c(V, \hat{p}) = -\int_{X \in V} p(X) \cdot \log_2\left(\frac{\hat{p}(X)}{p(X)}\right) dX, \tag{4}$$

where $\hat{p}$ denotes the probability density estimation function.

If we were to use the information provided by a bisection to modify the density distribution estimator $\hat{p}(X)$, we can utilize the difference between the cross entropy (Equation 4) of the cell and its two sub-cells (after bisection) as quality metric for selecting a bisection. A qualitative bisection yields a significant reduction of the cross entropy. More specifically, and according to Equation 5, a bisection $\mathcal{B}$ generating two volumes ($V_1$ and $V_2$) and two new density estimations ($\hat{p}_1$ and $\hat{p}_2$) would evaluate to the bisection entropy:

$$E_{\mathcal{B}} = E_c(V, \hat{p}) - E_c(V_1, \hat{p}_1) - E_c(V_2, \hat{p}_2). \tag{5}$$

Two terms remain to define. (1) $\hat{p}(X)$ is a representation of the density estimator available to the decoder. Since at each bisection, we only know the cardinality information of the cells, only a crude approximation can be computed. As our sampling model is uniform, we consider a constant density function on a cell derived from the volume of the cell and its cardinality. (2) The pointwise density $p(X)$ is more complex as it should represent the local density at point $X \in \mathbb{R}^3$ . We could consider a normal distribution around each point representing a "dilution" of this point on a volumetric domain. However, integrating many such distribution functions (one for each point) over non-trivial domains (polyhedral cells) is computationally intractable, and a simplified version of Equation 4 and 5 is necessary.

With the definition of $\hat{p}$ constant on a cell above, minimizing Equation 5 then amounts to minimize the variation of $p(X)$ over the sub-cells . Minimizing this variation for a sparse uniformly sampled point cloud mainly amounts to generating cells that contain points either almost everywhere or mostly nowhere. With this principle in mind, and matching our will to keep the computations tractable, we generated the final entropic metric to drive the bisections:

$$\boxed{\begin{aligned} Entropy = &-\frac{V_1}{V_t} \cdot \left(R_{1v} \cdot \log_2(R_{1v}) + R_{1f} \cdot \log_2(R_{1f})\right) \\ &-\frac{V_2}{V_t} \cdot \left(R_{2v} \cdot \log_2(R_{2v}) + R_{2f} \cdot \log_2(R_{2f})\right) \end{aligned} \quad (6)}$$

where $V_1$ denotes the volume of $\mathcal{C}_1$ (Cell 1), $V_t$ denotes he volume of $\mathcal{C}_1 \cup \mathcal{C}_2$, $R_{1f}$ denotes the ratio of "full" volume in $\mathcal{C}_1$ (see below), and $R_{1v}$ denotes the ratio of "empty" volume in $\mathcal{C}_1$ (see below).

This is a significant simplification of the previous equations. More specifically, we use three simplifications:

Firstly, and to estimate $p(X)$, we choose to model the density function around points as:

$$f(r) = \begin{cases} 1 & \text{if} \quad r < r_\varepsilon \\ 0 & \text{otherwise} \end{cases}$$

where $r_\varepsilon$ is chosen to be half the value of the estimated sampling distance calculated as:

$$r_\varepsilon = \frac{1}{n}\left(\sum_{p\in\mathcal{P}_c}\frac{1}{k}\left(\sum_{p_i\in\mathcal{N}_k(p)}\mathcal{L}_2(p,p_i)\right)\right),$$

where $\mathcal{P}_c$ denotes the input point cloud, $n$ denotes the number of points in $\mathcal{P}_c$, and $\mathcal{N}_k(p)$ denotes the k nearest neighbors of $p$ with $k = 6$.

Secondly, we ignore the intersections between balls to reduce computations, by summing up the ball volumes.

Lastly, we ignore the boundary conditions of the integration domain in Equation 3 and integrate over the whole space instead. The integral of $f$ over the whole space is $\frac{4}{3}.\pi.r_\varepsilon^3$, this integral for all points in a cell is used to represent the volume of "full" space. However, since this estimated volume could exceed the volume of the cell with the previous simplifications, we bound it to this volume to prevent degeneracies.

Finally, modifying Equation 3 yields the full volume estimation $V_f$ as:

$$V_f = \sum_{p\in\mathcal{C}}\gamma_f\cdot\left(\frac{4}{3}\cdot\pi\cdot r_\varepsilon^3\right) = n_\mathcal{C}\cdot\gamma_f\cdot\frac{4}{3}\cdot\pi\cdot r_\varepsilon^3,$$

where $\gamma_f$ is a volume correction term ( see C), $\mathcal{C}$ is the current cell, $r_\varepsilon$ is the sampling distance and $n_\mathcal{C}$ is the cardinality of $\mathcal{C}$. This approach is efficient to compute and remove the need for computing intersections between steps functions around each point of the input and the integration domain (cell $\mathcal{C}$, a polyhedral complex of arbitrary degree).

### 2.2.2. Normal direction quantification

Contrary to standard approaches relying on axis-aligned, symmetric planes, our bisecting planes must be encoded. The algebraic representation of a plane (Equation 7) is precise but encoding four floating point numbers would be prohibitive in terms of bitrates.

A 3D plane $\mathcal{P}$ is defined as:

$$\mathcal{P} = a\cdot x + b\cdot y + c\cdot z + d, \tag{7}$$

where $\vec{n}(a,b,c)$ denotes its normal vector, and $-d$ denotes the signed distance to the origin.

We design an adaptive quantification scheme that separates the plane into its normal direction (defined on the unit sphere $\mathcal{S}_2$ ) from the displacement, i.e. the signed distance from the origin (defined in $\mathbb{R}$). We sample $\mathcal{S}_2$ by sampling along a Fibonacci sampling sequence [MS01] (see B) and the displacement with uniform samling in the bound of the cell.

This approach scales to an arbitrary number of samples. However, a regular sampling is not optimal in few cases, especially when the bisected cell is anisotropic. More specifically, when the cell is pencil-shaped or flat, multiple bisecting directions sampled on the sphere are close or equivalent. When the cell is mostly planar, only planes orthogonal to the cell's "plane" are relevant for bisection. We thus only need to sample the space $\mathcal{S}_1$ of these orthogonal planes, or equivalently through a bijection $[0;2\cdot\pi]$ to obtain such a bisection plane. When the cell is pencil-shaped, all directions are equivalent except for degenerate cases where the plane direction and the major principal component of the cell are orthogonal. To optimize the sampling on $\mathcal{S}_2$, we use the principal component eigenvalues to estimate the cell dimension as:

$$\psi_1 = \sqrt[10]{\frac{12^2\cdot P_1^4}{P_2\cdot P_3}}, \tag{8}$$

where $P_{1,2,3}$ are the three eigenvalues of the principal component analysis of the volume of the cell.

$\psi_1$ is derived from the analysis of the dimensions of a box-shaped cell. More specifically, we consider a parallelepiped whose principal component coincide with the one of the cell. $\psi_1$ is set to match with the longest side of the parallelepiped. (similarly $\psi_2$ and $\psi_3$ are set to match respectively the second longest and smallest side).

We then compute the number of samples according to the cell dimensions with:

$$Q_o = \left(\lambda_o\cdot\frac{\sqrt{\psi_1\cdot\psi_2+\psi_1\cdot\psi_3+\psi_2\cdot\psi_3}}{r_\varepsilon}\right)^{dim-1} \tag{9}$$

| | |
|---|---|
| $\psi_1,\psi_2,\psi_3$ | $=$ Estimated cell dimension from eq. 8 |
| $r_\varepsilon$ | $=$ Point cloud sampling distance |
| $\lambda_o$ | $=$ Scaling parameter |
| $dim$ | $= \sqrt{\frac{(\psi_1+\psi_2+\psi_3)^3}{\psi_1^3+\psi_2^3+\psi_3^3}}$ |

We start by sampling the sphere with the aforementioned Fibonacci sequence, and we choose a selection of vectors that best match the cell geometry. More specifically, we use the dimension estimation along each principal component (Equation 8) to calculate an estimation of the local cell moment (Equation 10) over each sampled component. We calculate the sum $\sigma_t$ of the local moments for all components, and we divide the segment $[0,\sigma_t]$ into $k$ bins ($k$ being the range of quantified plane normal directions) $\Sigma_0 = \left\{\left[0,\frac{\sigma_t}{k}\right],\left]\frac{\sigma_t}{k},\frac{2.\sigma_t}{k}\right],\ldots,\left]\frac{(k-1).\sigma_t}{k},\sigma_t\right]\right\}$. More specifically, we calculate the cumulative sum on each component and locate the segment of $\Sigma_0$ that contains the said cumulative sum. If after the addition of the local moment, we end up in a different segment of $\Sigma_0$ than before, we keep the direction. Otherwise, we continue. We end up with a set of maximum $k$ quantized plane normal directions that are considered for the optimization.

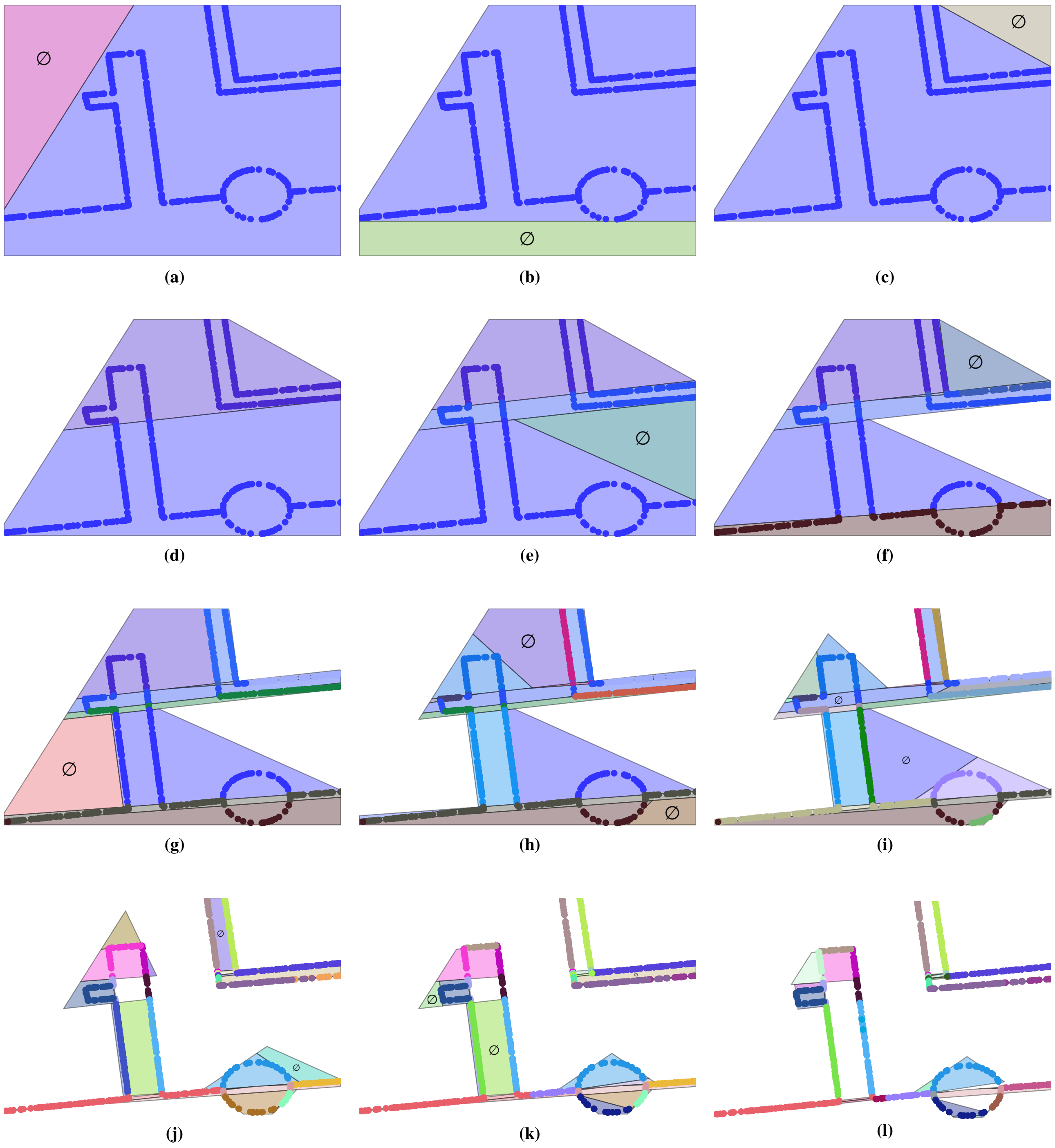


**Figure 6:** *Entropy-driven bisection in 2D. We recursively bisect the cells using bisecting lines that minimize the entropic metric defined by Equation 6. The optimizer first carves empty space (a, b, c). It then starts carving along sampled line segments (with high density) as this maximizes cardinality on one side while minimizing empty space. Note how the algorithm carves empty space again when a bisection generates a new opportunity to isolate a significant volume of empty space. The next bisection then carves empty space (e.g. (e) green cell or (g) pink cell).*

© 2024 The Authors.
Computer Graphics Forum published by Eurographics and John Wiley & Sons Ltd.

$$M(d_l) = M_v \cdot d_l^T \cdot \begin{pmatrix} \psi_2 \cdot \psi_3 \\ \psi_1 \cdot \psi_3 \\ \psi_1 \cdot \psi_2 \end{pmatrix}^T \tag{10}$$

$M_v$ = Matrix of PCA eigenvectors
$d_l$ = Current normalized direction vector
$\psi_1, \psi_2, \psi_3$ = Estimated cell dimension from eq. 8

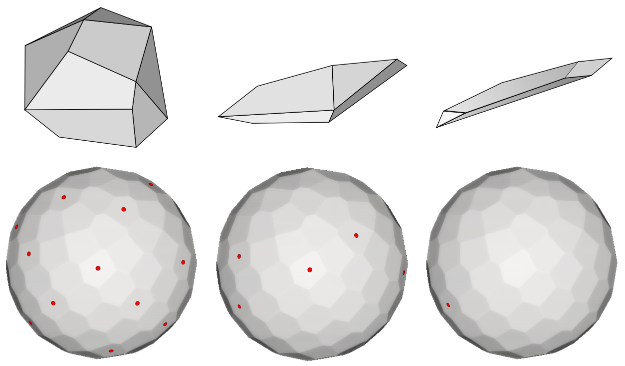


**Figure 7:** *Adaptive sampling of plane normal direction. A principal component analysis enables adaptive sampling of plane normal directions. The adaptative sampling variate the number of plane direction sampled. When the cell is 3D (Top-left) a high number of sample in $\mathcal{S}_2$ is used (Bottom-left). When it is plannar (Top middle) less sample directions are used and those tend to be orthogonal the the cell's "planar" direction (Bottom-middle). when the cell is "linear" (Top-right), only one direction, the principal component is used (Bottom-right)*

### 2.2.3. Displacement quantification

To quantize the plane displacement (distance to origin) we first select the maximal range of quantification (number of displacement values) based on the cell characteristics (PCA), the selected plane normal direction $d_i$ and the sampling distance $r_\varepsilon$. A user-defined parameter $\lambda_d$ (set by default to 0.6) enables variation in the quantization range with a maximum set by default to 128 bins.

$$Q_d = \lambda_d \cdot \frac{||M_v \cdot d^T \cdot M_p||}{r_\varepsilon} \tag{11}$$

$M_v$ = Matrix of PCA eigenvectors
$d$ = Current quantized plane direction (unit vector)
$M_p$ = Diagonal matrix of PCA eigenvalues

We then sample regularly between the two furthest points of the cell according to the current direction and range $Q_d$, see Figure 8.

### 2.2.4. Optimization

From the quantized plane normal direction and displacement discussed above, we obtain a total of $\sum_{i=1}^{Q_o} Q_d(d_i)$ candidate bisecting planes. For selecting the optimal bisecting plane, we proceed as follows. For each quantized plane normal direction, the $k$ different

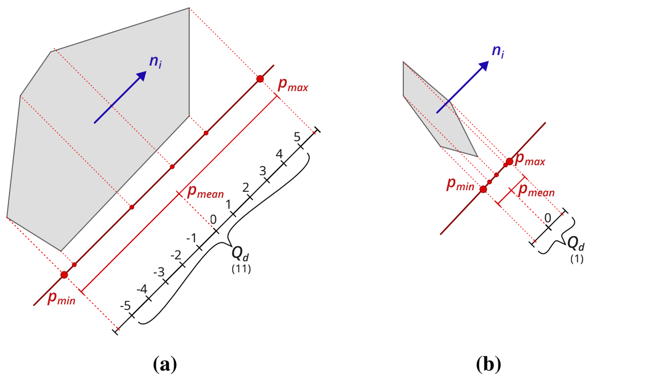


**Figure 8:** *Displacement sampling: $n_i$ denotes the current plane normal direction, $p_{min}$ and $p_{max}$ denote the min-max values projected onto component $n_i$, $p_{mean} = \frac{p_{min}+p_{max}}{2}$, $Q_d$ is the range of displacements. Left: The displacement quantization give multiple candidate (11) due to the cell dimension being important into the chosen direction. Right: The cell is small in the sampled direction so the only possibility for quantized direction is the displacement of the midpoint of the furthest opposed vertices for this direction.*

quantized displacement values subdivide the space into $k+1$ sections ($k$ parallel planes). Each input point in the current cell belongs to only one of the sections. We thus count the number of points in each section. For each quantized candidate displacement, we can efficiently determine the number of points in its positive and negative half spaces in order to evaluate the entropic metric (Equation 6) for each candidate plane (section 2.2.1). This procedure is repeated for each quantized plane normal direction, and the plane that minimize the entropic metric is selected for bisection. We then encode the quantized plane's normal direction and displacement into the bitstream.

We explored alternative optimization approaches such as gradient based method, gradient-free methods, and hierarchical optimization. However, the gradient-based approach revealed complex to optimize as many combinatorial changes occur when relocating the bisecting plane. The gradient-free approach based on the Nevergrad library [RT18] expects a notion of neighborhood of quantized values for better performance. However, this condition is not met for our quantified plane directions generated by Fibonacci sampling. Hierarchical optimization methods generated too many local minima to be of practical interest.

### 2.2.5. Prediction

For each cell $c$ to be bisected, we predict a default bisecting plane as follows. The normal of the bisecting plane is defined as the principal component of $c$, computed in closed form (see Appendix). Note that such a direction matches the one of the plane utilized in previous work [DG00]. In addition, the default plane is constrained to pass through the midpoint of the bounding segment (see Figure 8).

### 2.3. Cardinality

The cardinality (number of points) in one of the two sub-cells must be encoded after each bisection. The cardinality always refers to the sub-cell located in the halfspace above the bisecting plane. Predicting the cardinality of the sub-cell reveals difficult. A first issue comes from the variable range of possible values, even if they are bounded by the cardinality of the parent cell. A high range often corresponds to higher entropy since the probability of repeating values is small. Another issue is that, unlike previous work [DG00] where regular bisections tend to equaly spread points on both sides, our bisection does not promote equal volumes between the two sub-cells. In our experiments, a sub-cell cardinility is predicted with $\hat{n_c} = n_p * \frac{V_c}{V_p}$ where $\hat{n_c}$ is the child cell predicted cardinality, $n_p$ is the parent cell cardinality and $V_c(V_p)$ is the child (parent) cell volume. We then encode the difference between the predicted and the actual number of points on the positive side of the bisecting plane. The number of points in the negative side is deduced defined by matching the cardinality between the parent cell and its two sub-cells.

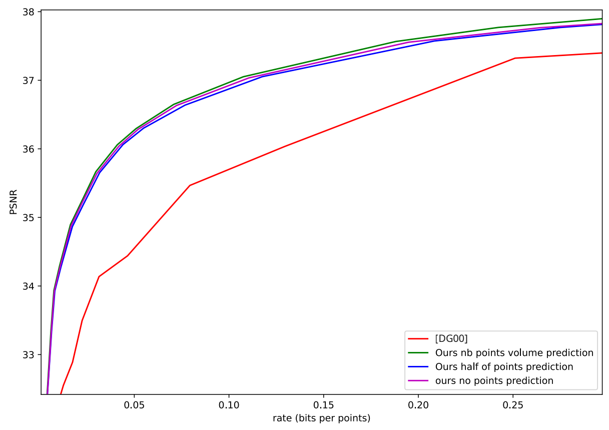


**Figure 9:** *Cardinality prediction. We plot the rate-distortion curves for three cardinality prediction methods: one based on volume share between the two cells, a second one predicting that each of the two sub-cells contains half of the points, the last one with no predictions. The baseline is also plotted for reference. We observe that the predictor based on volume share performs slightly better though the benefit is marginal.*

### 2.4. Ordering

Using a canonical (breadth-first) order (as in previous work) is suboptimal in our case because the depth of our bisection tree is in general higher than previous work. This is due to our entropy metric designed to isolate cells that are either empty or full. This leads to bisection operators carving empty space, leaving many dense sub-cells.

Figure 6 depicts how, after 3 bisections (Figures 6(a), 6(b), 6(c)) where we could have up to $2^3$ cells of interests, only one cell has non-zero cardinality.

In order to process the most informative cells first, we use a priority queue and define the priority value for cell $\mathcal{C}$ as $V_{\mathcal{C}} \cdot n_{\mathcal{C}}$ where $V_{\mathcal{C}}$ is the cell volume and $V_{\mathcal{C}}$ its cardinality. Figure 10 compares the rate-distortion curves for our method with two different ordering schemes (canonical vs importance-based), and the baseline approach [DG00] that utilizes a canonical ordering alternating plane orientations along x, y and z axes.

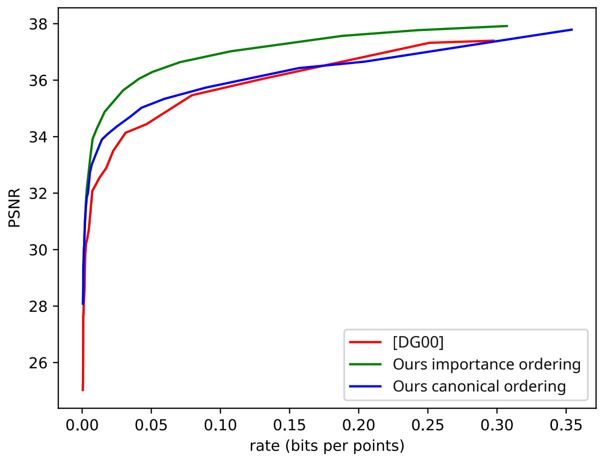


**Figure 10:** *Impact of ordering the bisection operators. We plot the rate-distortion curves for two ordering rules of the bisection operators: canonical vs importance-based, and for the baseline approach.*

### 2.5. Bitstream

The output bitstream first contains a header that describes the root cell (by default set to the bounding box) and the total cardinality. The rest of the bitstream is a series of sections that contains the information required to reconstruct the binary space partition. Each section encodes (1) the quantized residual of the plane normal direction, (2) the quantized residual of the plane displacement, and (3) the signed residual of the predicted cardinality. We utilize three distinct entropy encoders, each with its own context, and interlace the three related streams as a single stream. See Figure 11 and Figure 12.

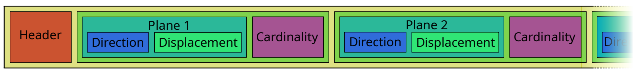


**Figure 11:** *Output bitstream. The header describes the root cell and total cardinality. Each section provides three elements: the bisecting plane direction and its displacement, and the cardinality.*

### 2.6. Decoding

The decoder proceeds like the encoder. It first reads the header, then starts decoding the bitstream via three arithmetic decoders. Each section is decoded and the bisecting planes and cardinality are reconstructed from the predictors. As all predictors are designed from the same (quantized) cells used during encoding, a similar state machine is preserved. At any point in time during decoding, a 3D point cloud can be reconstructed by uniformly sampling the cells.

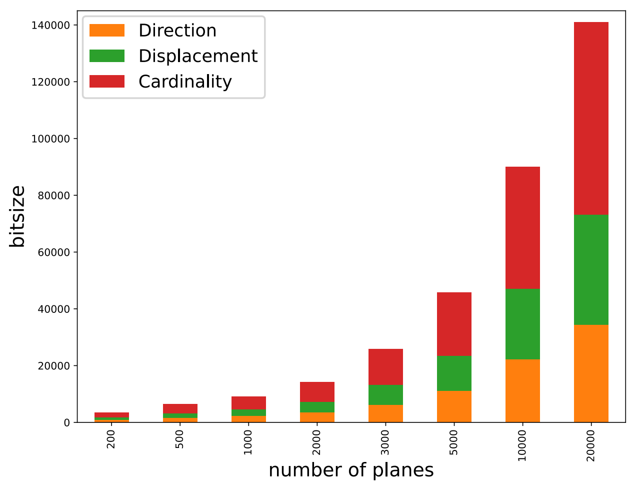


**Figure 12:** *Distribution in the bitstream. We plot the size of the different components of the stream: plane normal directions, plane displacements and cardinality. Encoding the planes is costly, but cardinality dominates the overall cost.*

## 3. Experiments

### 3.1. Implementation details

We implemented our approach in C++ using the CGAL library [The24], for the geometric data structures, the Eigen library for principal component analysis and other algebraic computations, the Intel TBB library for parallel computations, and the Draco library for both comparisons and arithmetic encoding. Additionally, we used the Nevergrad (Python) library [RT18] to conduct experiments on gradient-free optimization. All timings are measured on a Dell precision laptop with an Intel i7-12850HX CPU clocked at 2.4 GHz, and 64 GBytes of memory.

### 3.2. PSNR distortion

For measuring distortion, we adopt the common peak signal-to-noise ratio (PSNR) [TOF*17] computed point-to-point and symmetrically between the input point cloud and the point cloud reconstructed by the decoder. More specifically, the two-sided point-to-point distortion is defined as Chamfer distance:

$$C_D = \sum_{p\in\mathcal{P}_0} d_{\mathcal{L}_2}(p,\mathcal{N}_r(p)) + \sum_{p\in\mathcal{P}_r} d_{\mathcal{L}_2}(p,\mathcal{N}_0(p)), \qquad (12)$$

where $\mathcal{P}_0$ is The input point cloud; $\mathcal{P}_r$ is the Reconstructed/decoded point cloud, $\mathcal{N}_0$ is the Nearest neighbor point in $\mathcal{P}_0$, $\mathcal{N}_r$ is the Nearest neighbor point in $\mathcal{P}_r$, $d_{\mathcal{L}_2}$ is the L2 Euclidean distance

We then compute the PSNR as:

$$PSNR = 10\cdot\log\left(\frac{s^2}{C_D}\right), \qquad (13)$$

where $s$ denotes the longest edge diagonal of the bounding box (for other signals, $s$ denotes the representative intensity of the signal).

### 3.3. Evaluation and comparisons

We evaluate and compare our approach with three methods based on binary space partitions [DG00, Dra, BLL*06]. We reimplemented the baseline as a special case of our generic framework [DG00]. The Draco library [Dra] implements a slightly modified version of the baseline, where the reconstructed point cloud is obtained by generated one centroid per cell, instead of sampling the cells uniformly. We implemented a BSP method that uses a non-axis aligned subdivision scheme with a plane orthogonal to the local principal components of the points, and shifted to the median of the points [BLL*06].

For measuring rate-distortion tradeoffs for Draco [Dra], we interrupt the encoding at different subdivision depths. At each depth, we encode the stream obtained and measure its size to determine the bitrate. Given the current stream, the decoder reconstructs the point cloud and we measure the PSNR distortion.

This process generates a set of rate-distortion points for each approach. We plot the PSNR-distortion against the bitrate for the three following input 3D point clouds:

1. **Observatory:** LiDAR outdoor scan of the Eiffel observatory (city of Nice): 2M points, non-uniform sampling.
2. **Oil pump:** Laser scan of a mechanical part: 542k points sampled on a surface with semi-sharp features.
3. **Rhino:** Laser scan of a statue: 2M points sampled on a freeform surface [Jac78, Lar].

Those point clouds, as well as the results of progressive compression, are available in the additional ZIP archive.

Figures 14, 16, 15 plot rate-distortion curves. For each point cloud, we select three data points on the rate-distortion curve (①, ② and ③) with similar rates. For each data point, we depict the related reconstructed point cloud. Note that these data points are not exactly aligned vertically, as matching the exact same rate (abscissa) is difficult since the rate is known only when the interlaced streams are arithmetically encoded.

Our approach quickly captures the perceived geometry of the input 3D point cloud, and yields a more faithful approximation. In comparison, symmetric, axis-aligned approaches require many more planes and a higher bitrate to recover a similar reconstruction, which still exhibit significant voxelization artifacts. Although we spend more bits for encoding the bisecting planes, the rate-distortion curves and pointwise errors (Figure 17 and 18) show that the it is beneficial. Our general BSP data structure generates tighter cells that sandwich the points tightly on planar parts, and produce faceted approximations on curved parts. Note that our approach uses fewer and fewer quantized planes, down to the default planes (predicted at zero cost) as compression progresses (when cells become flat or pencil-like shaped). This prevents us from wasting bits where unnecessary. The additional material provides a ZIP archive containing an animated GIF of the progressive decompression of the Rhino point cloud.

### 3.4. Timings and memory

We now evaluate computation times and peak memory against number of input points and number of planes used for encod-

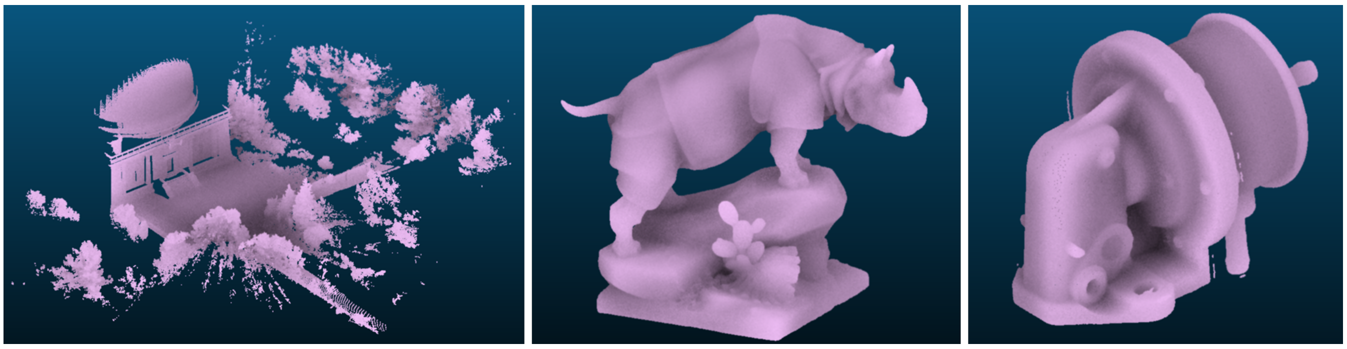


**Figure 13:** *Input 3D point clouds used for experiments and comparisons. Left: Observatory. Middle: Rhino. Right: Oil pump.*

| **Method** | **Ours** | | | **[Dg-00]** | | | **Draco** | | | **[BLL*06]** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Comp. rate** | **Size** | **PSNR** | **Comp. rate** | **Size** | **PSNR** | **Comp. rate** | **Size** | **PSNR** | **Comp. rate** | **Size** | **PSNR** |
| Observatory | 1:14,940 | 909B | 45.16 | 1:7,800 | 1.70kB | 41.08 | 1:11,598 | 1.14kB | 38.10 | 1:10262 | 1.29kb | 39.88 |
| | 1:1,916 | 6.92kB | 47.63 | 1:1,913 | 6.85kB | 43.62 | 1:1,242 | 10.68kB | 41.00 | 1:610 | 21.74kB | 42.80 |
| | 1:249 | 53.24kB | 50.24 | 1:153 | 85.60kB | 48.24 | 1:439 | 30.21kB | 46.97 | 1:128 | 102.56kB | 46.03 |
| Rhino | 1:26,291 | 913B | 49.81 | 1:11,561 | 1.87kB | 47.26 | 1:9,030 | 2.60kB | 44.50 | 1:9543 | 2.46kB | 46.91 |
| | 1:3,386 | 6.92kB | 54.23 | 1:3,023 | 7.75kB | 50.83 | 1:2,459 | 9.53kB | 47.29 | 1:2399 | 9.77kB | 49.02 |
| | 1:427 | 54.94kB | 57.71 | 1:225 | 103.77kB | 55.90 | 1:2,459 | 90.67kB | 53.72 | 1:200 | 116.33kB | 51.92 |
| Pump | 1:3,348 | 1.90kB | 23.15 | 1:3,834 | 1.66kB | 20.11 | 1:2,609 | 2.44kB | 16.30 | 1:2430 | 2.62kB | 17.02 |
| | 1:1,006 | 6.32kB | 25.48 | 1:953 | 6.67kB | 22.47 | 1:727 | 8.75kB | 19.39 | 1:667 | 9.53kB | 19.85 |
| | 1:142 | 44.74kB | 28.09 | 1:146 | 43.60kB | 25.91 | 1:97 | 65.59kB | 25.43 | 1:70 | 91.45kB | 26.00 |

**Table 1:** *Bitlengths and rate-distortion values. For three approaches and three input point clouds, we measure for three levels of details the compression rates, the bitlength after encoding and the corresponding PSNR distortion.*

| **#Points** | **#Planes** | **Timing** | | | **RAM** |
|---|---|---|---|---|---|
| | | **Opt.** | **Enc.** | **Dec.** | |
| 100k | 2k | 4s | 31ms | 64ms | 1.2GB |
| | 20k | 22s | 294ms | 685ms | |
| | 66.7k | 63s | 995ms | 2,281ms | |
| 200k | 2k | 10s | 27ms | 62ms | 2.9GB |
| | 20k | 58s | 275ms | 626ms | |
| | 122.8k | 178s | 2,003ms | 4,567ms | |
| 500k | 2k | 15s | 26ms | 59ms | 4.8GB |
| | 20k | 114s | 266ms | 609ms | |
| | 200k | 523s | 2,720ms | 6,232ms | |
| 2M | 2k | 22s | 27ms | 60ms | 12.5GB |
| | 20k | 148s | 255ms | 609ms | |
| | 200k | 601s | 3,957ms | 7,232ms | |

**Table 2:** *Timings and memory consumption on the observatory point cloud. We measure computation times for the three main steps: optimization (Opt.), encoding (Enc.) and decoding (Dec.), and peak memory consumption throughout all three steps.*

ing/decoding. We first generate four versions of the observatory point cloud via random point simplification, with 100k, 200k, 500k and 2M points. In Table 2, the optimization step includes the construction of the binary space partition (BSP) and the search for optimal planes (90% of the time).

Note how such a step consumes most of the computation times, even if it is performed offline. The peak memory consumption is indicated for the entire encoding/decoding sequence. In terms of efficiency, our decoder (online) is 15 times slower than the baselines, and the encoder (offline) is around three orders of magnitude slower mainly due to the search of optimal bisecting planes. Compared to the baselines [DG00, Dra], our prototype implementation one order of magnitude more memory-intensive, mainly due to the general BSP data structure that needs to represent the geometry of all cells instead of being implicit as in octrees or kD-trees. In addition, our BSP caches many intermediate geometric primitives and values used for encoding (plane equations, covariance matrices, cell volumes, etc.). Finally the memory consumption is measured for the full pipeline: BSP optimization, quantization, entropy coding, entropy decoding, dequantization, BSP reconstruction, point cloud reconstruction, and performance measurement. This pipeline contains a duplication of the BSP data structure (before compression and after reconstruction). For decompression scenarios in a production environment, only entropy decoding, dequantization, BSP re-

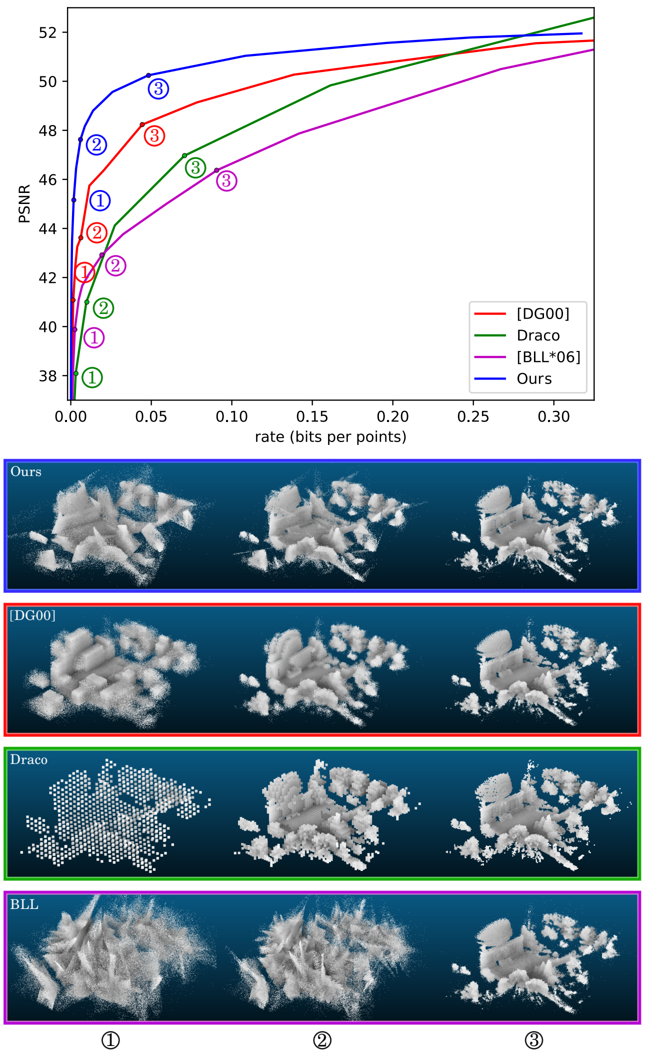


**Figure 14:** *Progressive compression of the observatory point cloud. Top: We plot the rate-distortion curves for three approaches. Our method is effective at decreasing distortion at very low bitrates. Bottom: we depict the reconstructed point clouds during decompression at three rate level for each method (marked on the top curve with ①, ② and ③). At low bitrates ①, our approach generates some outliers but the reconstruction already shows details of the geometric structure of the scene such as the path between the trees. These details are less visible with other approaches. For medium bitrates ②, the dome and stairs are identified in our approaches and do not appear in others. For high bitrates ③, our dome reconstruction exhibits less voxelization artifacts that other approaches.*

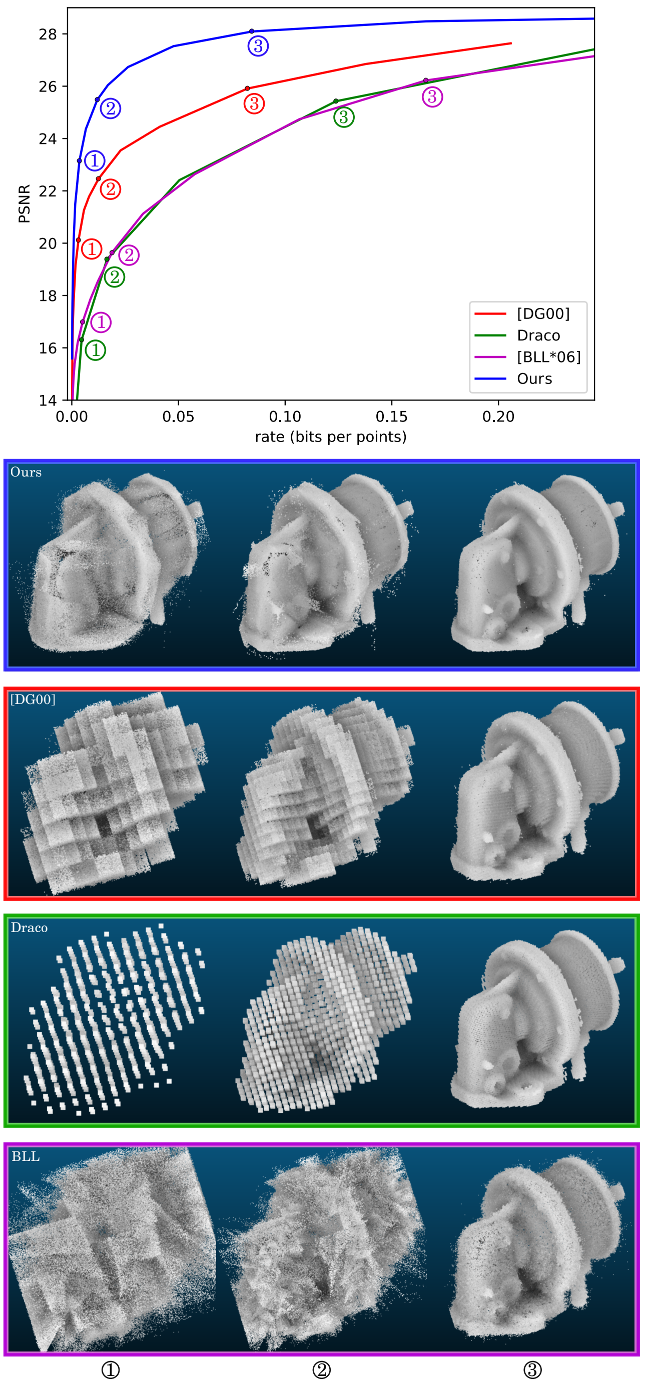


**Figure 15:** *Progressive compression of the oil pump point cloud. For lower bitrates ①, the general appearance and circular structures of the pump are already present in our reconstruction while it is still vague for other methods. The cylindrical parts and the base are already present. For medium bitrates ②, our approach generates few outliers but much fewer voxel artifacts compared to the baselines. A similar observation holds For higher bitrates ③.*

construction and point cloud reconstruction is necessary and in this case the memory usage is mostly impacted by the BSP structure 1.8GB for 2M points. We expect that an optimized version would further reduce memory consumption .

## 4. Conclusion

We introduced a new framework for progressive compression of 3D point clouds, based on general binary space partitioning (i.e., via asymmetric and non axis-aligned planar bisections). The construction of the said partition is driven by a novel entropy criterion

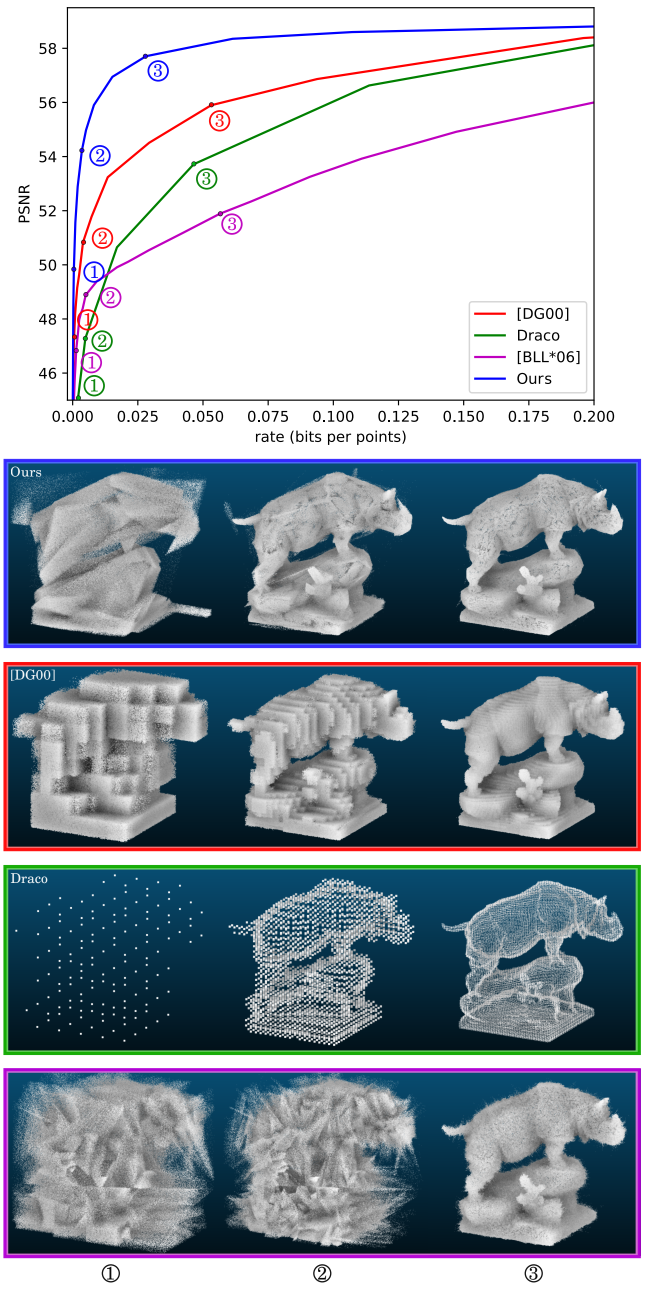


**Figure 16:** *Progressive compression of the rhino point cloud. For low bitrates ①, our method generates outliers but the perceived topology such as the tunnel between the rock and belly of the rhino is already better recovered than the baselines. In addition, the rock stand is already well oriented and its geometry better approximated. For medium bitrates ②, our method already recovers the horns and ears of the rhino and better separates the plant from the ground. For high bitrates ③, the point cloud is faithfully reconstructed for all approaches, but many aliasing artifacts are visible on the baselines due to the axis-aligned BSP data structure. These artifacts are especially visible on the back, tail, horns and ears of the rhino.*

that favors the generation of cells of the partition that are either full or empty. For each cell subject to bisection, we perform an adaptive quantization of the space of candidate bisecting planes, and search for the optimal plane within the discrete solution space. A principal component analysis of the cell provides us with a means to adapt the above quantization process, and to determine the default (predicted) bisecting plane. The output bitstream is mainly generated by interlacing two types of encoded information: the bisecting planes and the cardinality of the cells. While encoding the bisecting planes has a cost compared to previous work that rely upon implicit (axis-aligned and symmetric) planes [DG00], our experiments show that it provides a clear benefit in terms of rate-distortion tradeoff and voxelization artifacts, especially for low bitrates where compression is notoriously difficult.

**Limitations.** One limitation of our approach lies in the discrete solver for finding the optimal bisecting planes. Despite our efforts to explore gradient-descent, hierarchical or gradient-free optimization, our experiments revealed that brute-force optimization yields the best trade-offs between quality of the solutions (value of the objective function) and computation times. Another limitation comes from the simplification made to accelerate the evaluation of the entropy criterion used for bisecting. While such a simplification operates fine in most cases, refining it for cells that exhibit a very non-uniform sampling would improve accuracy. Lastly since our entropic metric is based on an estimation of local point cloud density, it could under-perform in case of non-uniform point sampling. though we still have interesting performances for the observatory point cloud (a non-uniformly sampled LIDAR point cloud) the results could vary with a more aggressive disparity in the sampling uniformity.

**Future work.** In future work, we wish to explore other optimization methods to find optimal bisecting planes in shorter times. We also wish to replace the current default root cell (bounding box) with a tighter convex polyhedron of arbitrary degree, but determining the optimal degree (and related polyhedron) that would yield better rate-distortion curves is still an open problem. Finally, we wish to explore non-uniform sampling processes for the cells during decoding. Our current approach resorts to uniform sampling to reflect the available information for a single cell. However, since we know the cardinality of all cells of the partitions, one possible direction would be to use machine learning for regressing a probability density function from the current context available for the decoder (BSP partition and cardinality information). Transmitting additional information about the density function would be a valid option for reducing distortion and possibly better predict bisections. Nevertheless, exploring the trade-off between (lossy) encoding of such a PDF and reducing distortion would be another significant challenge.

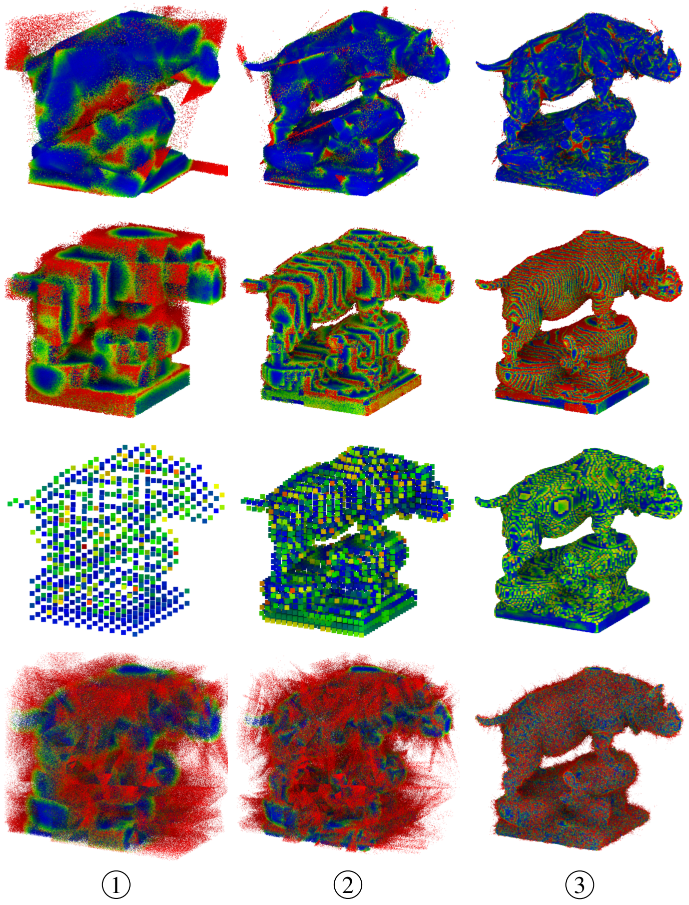


**Figure 17:** *Pointwise distortion. We visually compare the point-to-point distance with the baselines using a rainbow color ramp (red = large, blue = small error) and a common range, for different bitrates. Top: Our method. Middle-Top row: [dg-00]. Middle-Bottom: Draco. Bottom: [BLL*06] Left: Low bitrate. Middle: medium bitrate. Right: High bitrate. Although our approach tends to generate more outlier noise, its tight "sandwiching" capability yields a more faithful reconstruction, as already confirmed by the rate-distortion curves.*

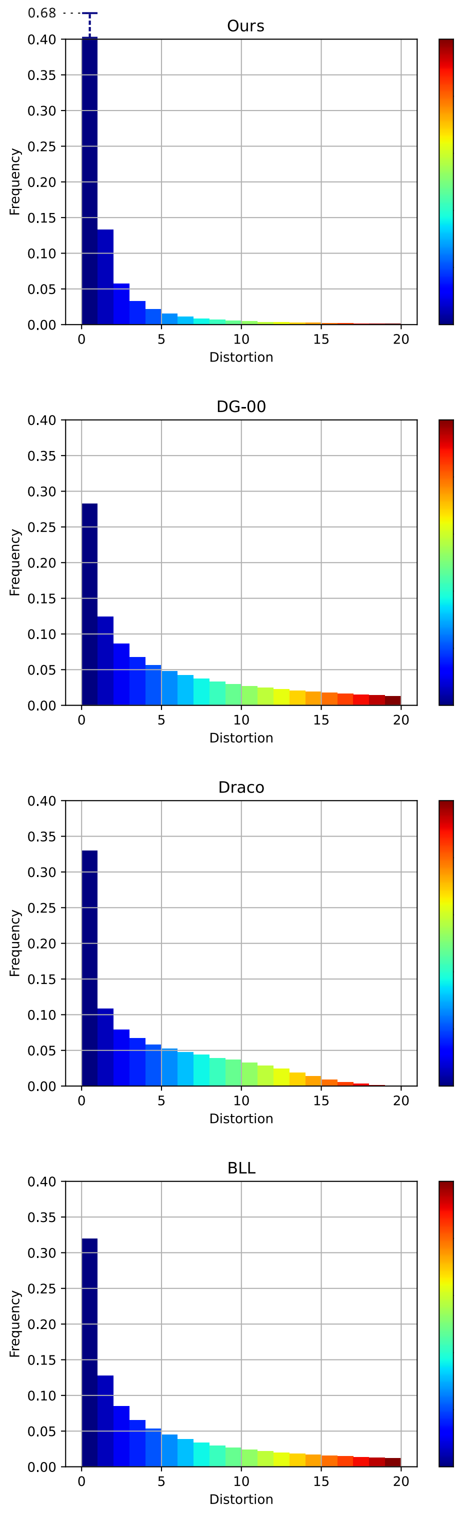


**Figure 18:** *Distribution of distortion. We plot the distribution of the two-sided point-to-point $\mathcal{L}_2$ error metric for the three approaches on the rhino point cloud (highest resolution, ③) from Figure 17. The same scale is applied for each plot, Note that for our method, the first column peaks at 68% but is cropped for improved readability of all graphs.*

## References

[BLL*06] Bordignon A., Lewiner T., Lopes H., Tavares G., Castro R.: Point set compression through bsp quantization. In *2006 19th Brazilian Symposium on Computer Graphics and Image Processing* (2006), IEEE, pp. 229–238. 2, 10

[CPZ19] Cao C., Preda M., Zaharia T.: 3d point cloud compression: A survey. In *The 24th International Conference on 3D Web Technology* (2019), pp. 1–9. 2

[DG00] Devillers O., Gandoin P.-M.: Geometric compression for interactive transmission. In *Proceedings of Visualization 2000* (2000), IEEE, pp. 319–326. 2, 3, 4, 8, 9, 10, 11, 13

[DQC16] De Queiroz R. L., Chou P. A.: Compression of 3d point clouds using a region-adaptive hierarchical transform. *IEEE Transactions on Image Processing 25*, 8 (2016), 3947–3956. 3

[Dra] Draco point cloud compression. URL: https://google.github.io/draco/. 3, 10, 11

[GAP08] Gupta A., Alliez P., Pion S.: *Principal component analysis in CGAL*. Tech. rep., Inria, 2008. 16

[GD02] Gandoin P.-M., Devillers O.: Progressive lossless compression of arbitrary simplicial complexes. *ACM Transactions on Graphics (TOG) 21*, 3 (2002), 372–379. 3

[GKIS05] Gumhold S., Karni Z., Isenburg M., Seidel H.-P.: Predictive point-cloud compression. In *SIGGRAPH 2005 Sketches*. ACM, 2005, pp. 137–es. 2

[Gra84] Gray R.: Vector quantization. *IEEE Assp Magazine 1*, 2 (1984), 4–29. 2

[GRR*22] Guarda A. F., Rodrigues N. M., Ruivo M., Coelho L., Seleem A., Pereira F.: It/ist/ipleiria response to the call for proposals on jpeg pleno point cloud coding. *arXiv preprint arXiv:2208.02716* (2022). 3

[Haa09] Haar A.: *Zur theorie der orthogonalen funktionensysteme*. Georg-August-Universitat, Gottingen, 1909. 2

[HPKG06] Huang Y., Peng J., Kuo C.-C. J., Gopi M.: Octree-based progressive geometry coding of point clouds. In *PBG@ SIGGRAPH* (2006), pp. 103–110. 2

[Huf52] Huffman D. A.: A method for the construction of minimum-redundancy codes. *Proceedings of the IRE 40*, 9 (1952), 1098–1101. 2

[ILS05] Isenburg M., Lindstrom P., Snoeyink J.: Lossless compression of predicted floating-point geometry. *Computer-Aided Design 37*, 8 (2005), 869–877. 2

[ISO12] ISO F. P. B.: International organization for standardization organisation internationale de normalisation iso/iec jtc1/sc29/wg11 coding of moving pictures and associated audio, 2012. 2

[Jac78] Jacquemart H.-A.: Rhinocéros, 1878. Courtesy of "Dépot des sculptures de la ville de Paris. URL: https://threedscans.com/depot-des-sculptures-de-la-ville-de-paris/rhino/. 10

[JDL*21] Janus S., Das B., Labbe H., Oh J. D., Cilingir G., Holland J., Biswal N., Chiu Y.-J., Xu Q., Varerkar M., et al.: Point cloud viewpoint and scalable compression/decompression, June 29 2021. US Patent 11,049,266. 3

[JPM*19] Jang E. S., Preda M., Mammou K., Tourapis A. M., Kim J., Graziosi D. B., Rhyu S., Budagavi M.: Video-based point-cloud-compression standard in mpeg: From evidence collection to committee draft [standards in a nutshell]. *IEEE Signal Processing Magazine 36*, 3 (2019), 118–123. 3

[JTC*12] Jiang W., Tian J., Cai K., Zhang F., Luo T.: Tangent-plane-continuity maximization based 3d point compression. In *2012 19th IEEE International Conference on Image Processing* (2012), IEEE, pp. 1277–1280. 2

[KLL*18] Kathariya B., Li L., Li Z., Alvarez J., Chen J.: Scalable point cloud geometry coding with binary tree embedded quadtree. In *2018 IEEE International Conference on Multimedia and Expo (ICME)* (2018), IEEE, pp. 1–6. 2

[Lar] Laric O.: Three d scans. URL: https://threedscans.com. 10

[Mal89] Mallat S. G.: A theory for multiresolution signal decomposition: the wavelet representation. *IEEE transactions on pattern analysis and machine intelligence 11*, 7 (1989), 674–693. 2

[MS01] Mojsilovic A., Soljanin E.: Color quantization and processing by fibonacci lattices. *IEEE transactions on image processing 10*, 11 (2001), 1712–1725. 6

[MSMW07] Merkle P., Smolic A., Muller K., Wiegand T.: Multi-view video plus depth representation and coding. In *2007 IEEE International Conference on Image Processing* (2007), vol. 1, IEEE, pp. I–201. 2

[MTS*22] Mammou K., Tourapis A., Su Y., Jungsun K., Robinet F. A., Valentin V. G.: Occupancy map block-to-patch information compression, June 21 2022. US Patent 11,367,224. 3

[NQVD21] Nguyen D. T., Quach M., Valenzise G., Duhamel P.: Multiscale deep context modeling for lossless point cloud geometry compression. In *2021 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)* (2021), IEEE, pp. 1–6. 3

[OS04] Ochotta T., Saupe D.: Compression of point-based 3d models by shape-adaptive wavelet coding of multi-height fields. In *Proceedings of the First Eurographics Conference on Point-Based Graphics* (2004), SPBG'04, Eurographics Association, p. 103–112. 2

[PG01] Pauly M., Gross M.: Spectral processing of point-sampled geometry. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), pp. 379–386. 2, 3

[PK03] Peng J., Kuo C. J.: Octree-based progressive geometry encoder. In *Internet Multimedia Management Systems IV* (2003), vol. 5242, SPIE, pp. 301–311. 2

[QLX21] Que Z., Lu G., Xu D.: Voxelcontext-net: An octree based framework for point cloud compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 6042–6051. 2, 3

[QVD19] Quach M., Valenzise G., Dufaux F.: Learning convolutional transforms for lossy point cloud geometry compression. In *2019 IEEE international conference on image processing (ICIP)* (2019), IEEE, pp. 4320–4324. 3

[RT18] Rapin J., Teytaud O.: Nevergrad - A gradient-free optimization platform. https://GitHub.com/FacebookResearch/Nevergrad, 2018. 8, 10

[Sha48] Shannon C. E.: A mathematical theory of communication. *The Bell system technical journal 27*, 3 (1948), 379–423. 5

[SK06] Schnabel R., Klein R.: Octree-based point-cloud compression. *Point-based graphics at ACM SIGGRAPH 3* (2006), 111–121. 2

[SPB*18] Schwarz S., Preda M., Baroncini V., Budagavi M., Cesar P., Chou P. A., Cohen R. A., Krivokuća M., Lasserre S., Li Z., et al.: Emerging mpeg standards for point cloud compression. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems 9*, 1 (2018), 133–148. 2

[The24] The CGAL Project: *CGAL User and Reference Manual*, 5.6.1 ed. CGAL Editorial Board, 2024. URL: https://doc.cgal.org/5.6.1/Manual/packages.html. 9

[TOF*17] Tian D., Ochimizu H., Feng C., Cohen R., Vetro A.: Geometric distortion metrics for point cloud compression. In *2017 IEEE International Conference on Image Processing (ICIP)* (2017), IEEE, pp. 3460–3464. 10

[TTCT19] Tu C., Takeuchi E., Carballo A., Takeda K.: Point cloud compression for 3d lidar sensor using recurrent neural network with residual blocks. In *2019 International Conference on Robotics and Automation (ICRA)* (2019), IEEE, pp. 3274–3280. 3

[TTMT16] Tu C., Takeuchi E., Miyajima C., Takeda K.: Compressing continuous point cloud data using image compression methods. In *2016 IEEE 19th international conference on intelligent transportation systems (ITSC)* (2016), IEEE, pp. 1712–1719. 3

[WLHS23] Wang C., Lasang P., Han C. D., Sugio T.: Three-dimensional data encoding method, three-dimensional data decoding method, three-dimensional data encoding device, and three-dimensional data decoding device, Apr. 11 2023. US Patent 11,625,865. 3

[WNC87] Witten I. H., Neal R. M., Cleary J. G.: Arithmetic coding for data compression. *Communications of the ACM 30*, 6 (1987), 520–540. 4

[ZFL14] Zhang C., Florencio D., Loop C.: Point cloud attribute compression with graph transform. In *2014 IEEE International Conference on Image Processing (ICIP)* (2014), IEEE, pp. 2066–2070. 3

[ZGL20] Zhang X., Gao W., Liu S.: Implicit geometry partition for point cloud compression. In *2020 Data Compression Conference (DCC)* (2020), IEEE, pp. 73–82. 2

[ZXLL17] Zhu W., Xu Y., Li L., Li Z.: Lossless point cloud geometry compression via binary tree partition and intra prediction. In *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)* (2017), IEEE, pp. 1–6. 2

## A. Covariance matrix of a polyhedron

Our method repeatedly requires to estimate the dimensions of convex 3D polyhedral cells along three mutually orthogonal axes. We perform a principal component analysis by computing the covariance matrix of the cells in closed form and its eigenvalues [GAP08]. More specifically, we decompose each convex polyhedron cell into a set of tetrahedra (by staring from an arbitrary vertex) and compute its covariance matrix as follows:

for a set of $n$ Tetrahedron composed of the four vertices

$$\mathcal{V}_i = \{v_1(x_1,y_1,z_1), v_2(x_2,y_2,z_2), v_3(x_3,y_3,z_3), v_4(x_4,y_4,z_4)\}$$

$$\begin{aligned} \mathrm{C}_P = &\sum_{i=1}^{n} \left[ |A_s| A_{s_i} \mathrm{M}_0 A_{s_i}^T + m_{s_i} \left( c_{s_i} V_{s_i}^T + V_{s_i} c_{s_i}^T - V_{s_i} V_{s_i}^T \right) \right] \\ &- \frac{1}{(\sum_{i=1}^{n} m_{s_i} c_{s_i})} \left( \sum_{i=1}^{n} m_{s_i} c_{s_i} \right) \left( \sum_{i=1}^{n} m_{s_{s_i}} c_{s_i}^T \right) \end{aligned} \tag{14}$$

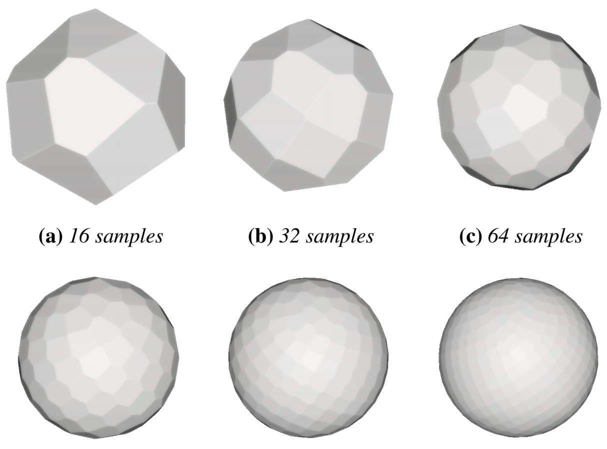
$$\begin{aligned} M_0 &= \begin{bmatrix} 1/60 & 1/120 & 1/120 \\ 1/120 & 1/60 & 1/120 \\ 1/120 & 1/120 & 1/60 \end{bmatrix} \\ A_{s_i} &= \begin{bmatrix} x_2 - x_1 & x_3 - x_1 & x_4 - x_1 \\ y_2 - y_1 & y_3 - y_1 & y_4 - y_1 \\ z_2 - z_1 & z_3 - z_1 & z_4 - z_1 \end{bmatrix} \\ V_{s_i} &= [x_1, y_1, z_1]^T \end{aligned}$$

## B. Fibonacci sampling

The Fibonacci unit sphere sampling generates a serie of points sampled on the unit sphere $\mathcal{S}_2$. This approach uniformly samples the $z$ axis and the angle around the $z$ axis to generate a set of points on $\mathcal{S}_2$ :

$$\begin{aligned} \mathcal{S}_f = \{(x,y,z) |\ & z = 1 - \frac{k}{n-1}, \\ & x = arcos(z) \cdot cos(k * \theta_g), \\ & y = arcos(z) \cdot sin(k * \theta_g), \end{aligned}$$

where $n$ denotes the number of samples, $k \in [\![0, n-1]\!]$ and $\theta_g$ denotes the golden angle $\pi.(3-\sqrt{5})$.


**(a)** *16 samples* **(b)** *32 samples* **(c)** *64 samples*

**(d)** *128 samples* **(e)** *256 samples* **(f)** *512 samples*

**Figure 19:** *Fibonacci sphere sampling. Each facet normal corresponds to a sampled normal direction from the Fibonacci sequence. The figure depicts a full sphere but we quantize only the half-sphere in practice.*

## C. Volume correction.

When a cell become skinny, considering the entire volume of a ball per point rapidly "fills" its space, limiting the quality of the entropy metric. We thus use a volume correction term based on the cell dimensions estimated from volumetric principal component analysis computed in closed form (see Annex). More specifically, eigenvalues of the cell's covariance matrix enable estimating the local cell dimensions along the pca axes $(\psi_1, \psi_2, \psi_3)$ (Equation 8):

where $P_{1,2,3}$ are the three eigenvalues of the principal component analysis of the volume of the cell.

For each dimension value $\psi_d$, we consider the volume of the ball cropped by two symmetrically placed planes at distance $\psi_d$ from the ball center. This way, the volume of the ball is reduced when one or several dimensions of the cell are smaller than the sampling distance $r_\varepsilon$.

We then define the volume correction for one dimension $R_d$ as:

$$R_d = \min\left(1,\ 1 - \frac{1}{2} \cdot \left( \frac{\psi_d}{r_\varepsilon}^3 - 3 \cdot \frac{\psi_d}{r_\varepsilon} + 2 \right) \right), \tag{15}$$

and the full volume correction $R_f$ as:

$$R_f = R_1 \cdot R_2 \cdot R_3. \tag{16}$$

The above correction enables the entropic metric to remain reliable when a cell dimension become too small (flat cell or pencil-like cell).

Figure 6 depicts a recursive bisection sequence in 2D guided by the entropic metric (eq. 6). We observe that this approach tends to favor bisections that remove empty space or maximize homogeneity of point distributions in sub-cells.