A polyhedral reconstruction of a 3D object from a chain code and a low-density point cloud

Osvaldo A. Tapia-Dueñas[1], Hiram H. López[2*], Hermilo Sánchez-Cruz[3]

[1]Department of Mathematics, Computer Science, and Data Science, John Carroll University, University Heights, OH, USA.

[2*]Department of Mathematics, Virginia Tech, Blacksburg, VA, USA.

[3]Departamento de Ciencias de la Computación, Universidad Autónoma de Aguascalientes, Aguascalientes, Aguascalientes, México.

*Corresponding author(s). E-mail(s): hhlopez@vt.edu;
Contributing authors: otapiaduenas@jcu.edu; hermilo.sanchez@edu.uaa.mx;

Abstract

The manipulation of 3D objects is becoming crucial for many applications, such as health, industry, or entertainment, to mention some. However, these 3D objects require substantial energy and different types of resources. With the goal of obtaining a simplified representation of a 3D object that can be easily managed, for example, for transmission, in some recent works, the authors associate low-density point clouds with a 3D object that simplifies the original 3D object. More precisely, given a 3D object in a polyhedral format, some authors associate a chain code and then use grammar-free context to obtain key points that give rise to several point clouds with different densities. In this work, we complete the cycle by developing a polyhedral reconstruction from an associated low-density point cloud and the chain code. The polyhedral reconstruction is crucial for handling 3D objects because it allows us to visualize them after they are efficiently compressed and transmitted. We apply our algorithms to well-known 3D objects in the literature. We use the Hausdorff and Chamfer distances to compare our results with the state-of-the-art proposals. We show how our proposed polyhedral reconstruction based on a helical chain code reconstructs a medical image represented or transmitted by slices into a 3D object in a polyhedral format, helping thus to mitigate and alleviate the management of 3D medical objects. The polyhedron that we propose provides better compression when compared with the original set of slices of a 3D medical object.

Keywords: Point cloud, Triangulation, 3D Shape, Data reduction, Polyhedral approximation, 3D computed tomography

1 Introduction

Three-dimensional (3D) image analysis and processing, as recognition, reconstruction, and optimal storage, has significant importance in various fields and applications, such as health, industry, and entertainment, or to extract useful information from three-dimensional scenes, to mention some. Some of the critical reasons for their importance are artificial vision and robotics [1], virtual and augmented reality [2, 3], scene reconstruction, and object detection and facial recognition [4].

*Hiram H. López was partially supported by the NSF grants DMS-2201094 and DMS-2401558. Hermilo Sánchez-Cruz was partially supported by Universidad Autónoma de Aguascalientes under grant PII24-4. Part of this work was done during the visit of Osvaldo Tapia-Dueñas to Hiram H. López at Virginia Tech.

Among efforts to achieve the analysis, processing, and representation of 3D objects, some methods without loss of information developed in the literature are [5–7]. In the recent work *3D object simplification using chain code-based point clouds* [8], the authors simplify a 3D object to a point cloud through a chain code. To be more precise, given a 3D object in a polyhedral format, for example, in a PLY format, the authors in [8] associate a sequence of point clouds of different densities that help the user decide between the trade-off size versus simplification. The sequence of the point clouds depends on a chain code and a helical path associated with the original 3D object. In this work, we propose reconstructing the 3D object in a polyhedral format from a point cloud. The polyhedral reconstruction is crucial for handling 3D objects because it allows us to visualize them after they are efficiently compressed and transmitted. We apply our algorithms to well-known 3D objects in the literature. Finally, we show how our proposed polyhedral reconstruction based on a chain code helps mitigate and alleviate the management of medical objects. The diagram in Fig. 1 represents where this paper is situated with respect to the state of the art to manage 3D objects using chain codes.

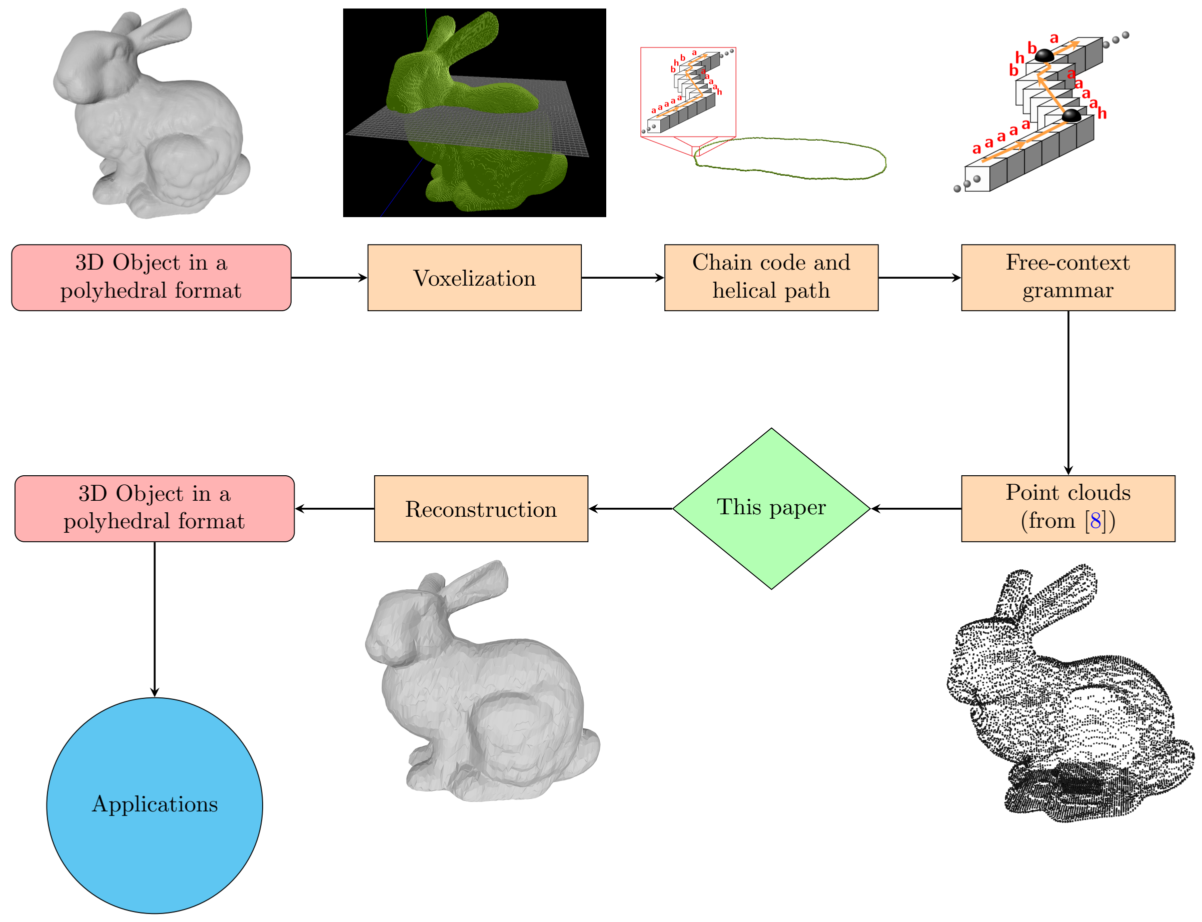

3D Object in a polyhedral format
Voxelization
Chain code and helical path
Free-context grammar
Point clouds (from [8])
This paper
Reconstruction
3D Object in a polyhedral format
Applications

Fig. 1 This paper reconstructs the 3D object in a polyhedral format from a point cloud.

There are other methods to reconstruct simplified 3D point clouds in the literature. In [9], the authors use optimized distance functions to reconstruct the 3D object while preserving the topology. The work [10] presents a method to obtain simplified and uniform resample points for surface reconstruction. In [11], the authors present new algorithms for global shape reconstructions from sparse tracked surface points. In order to achieve 3D reconstruction from multiple layers, in [12], the authors propose a deep learning architecture called Mesh Reconstruction Network (MR-NET) that is applied in real-time despite missing data and sparse annotations, reaching accurate shape reconstruction in the presence of incomplete or noisy contours. In [13], the authors use a sphere as a guide surface to obtain the surface reconstruction from a point cloud. To reconstruct a 3D mesh that shows detailed characteristics of the objects, the authors of [14] use the set of flat points progressively to avoid massive and quantitative calculations.

The strategy we follow in this article differs from other literature approaches because we aim to reconstruct the 3D object from the point clouds with the help of a helical chain code. We highly rely on the point clouds having an associated chain code. Even when the point clouds are enough to visualize the shape of the object roughly, the helical chain code plays a crucial role in reconstructing the 3D object because the code preserves the order of the original voxels. For example, our method differs from those based on machine learning [15, 16] because our proposal already includes the shape features (chain codes and point coordinates) to be used for the image representation, which avoids any other type of search or learning by any model.

In the health area, the 3D image representation and analysis has found meaningful applications in medicine and diagnosis [17], surgical planning, medical education, volumetric measurements [18], and Digital Imaging and Communications in Medicine (DICOM) files to optimally store the thousands of slices that tomographic medical imaging systems produce or for the display, storage, and transmission of anatomical images for 3D model creation [19]. We show how our proposed polyhedral reconstruction based on a helical chain code reconstructs a medical image represented or transmitted by slices into a 3D object in a polyhedral format, helping to mitigate and alleviate the management of 3D medical objects.

The diagram in Fig. 2 represents what this paper is doing with respect to the reconstruction of 3D medical objects using chain codes.

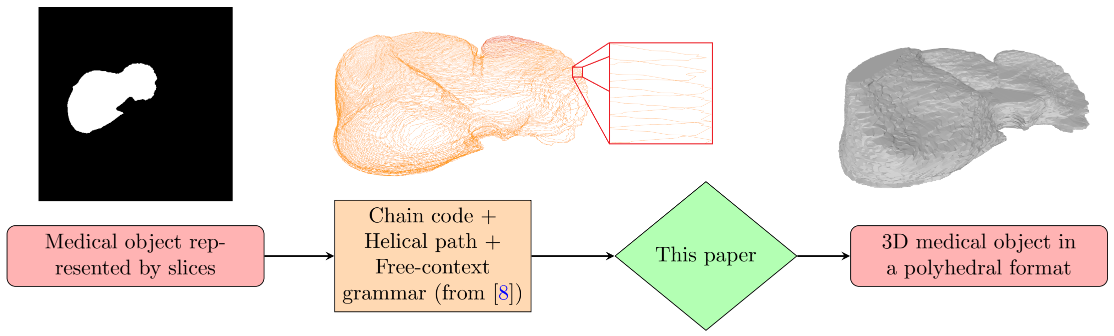

Medical object represented by slices
Chain code + Helical path + Free-context grammar (from [8])
This paper
3D medical object in a polyhedral format

Fig. 2 This paper reconstructs the 3D medical object in a polyhedral format from the set of slices.

We follow this article in the following way. In Section 2, we give some definitions, concepts, and procedures that we use throughout this paper, including how to obtain a helical chain code and a point cloud from a 3D object. In Section 3, we present an algorithm to reconstruct a 3D object in a polyhedral format from a point cloud and their respective helical chain code. In Section 4, we present the results and analysis of our proposed reconstructive algorithm. We show in Section 5 how our proposed polyhedral reconstruction based on a helical chain code reconstructs 3D medical images represented or transmitted by slices, helping thus to mitigate and alleviate the management of medical objects. Finally, in Section 6, we give some conclusions and further work.

2 Preliminaries

In [8], the authors present a method to obtain a chain code and a low-density point cloud from a 3D object. Among the main goals of the chain code and the low-density point cloud is to obtain a simplified representation of the 3D object that can be easily managed, for example, for compression and transmission. As our primary goal in this paper is to reconstruct the 3D object in a polyhedral format from the point cloud and the chain code, in this section, we present a summary of [8] to obtain the chain code and the low-density point clouds from the original 3D object. For a more detailed explanation, see [8]. In addition, we present a helical chain code that helps us keep order in the voxels for reconstruction. For a more detailed explanation of helical chain codes, see [5] and the references therein.

A *chain code* is a common and compact way to represent a two-dimensional (2D) object. The *Freeman chain code* (F8), proposed by Freeman in 1961 [20], is a sequence of symbols that belong to the set $\{0, 1, 2, 3, 4, 5, 6, 7\}$ that codifies the information of the contour shape of a 2D object. Every symbol represents two adjacent pixels (see Fig. 3).

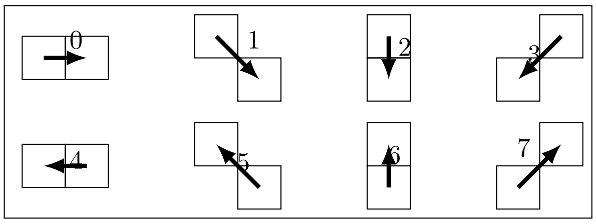


Fig. 3 Symbols to encode with $F8$.

The *angle Freeman chain code*, based on the chain code $F8$ and proposed by Kui and Žalik in 2005 [21], is a sequence of symbols that belong to the set $\{a, b, c, d, e, f, g, h\}$ and codifies the angles of the contour shape of the 2D object (see Fig. 4). The symbol e arises when a pixel in the contour shape of $\mathcal{B}$ is adjacent to only one pixel of the contour shape.

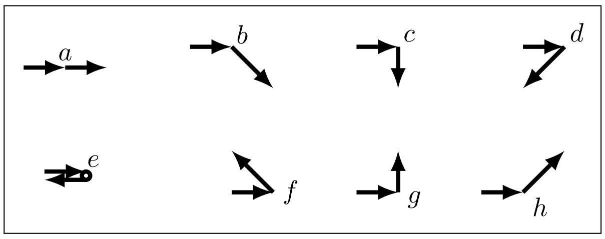


Fig. 4 Symbols to encode with $AF8$.

Example 2.1. Consider the 2D object $\mathcal{B}$ shown in Fig. 5. If we start to encode from the top left-most pixel, we obtain that the chain codes are $F8\,(\mathcal{B})$ = 00022123444460075467 and $AF8\,(\mathcal{B})$ = *baacahbbbaaaaccahghcb*.

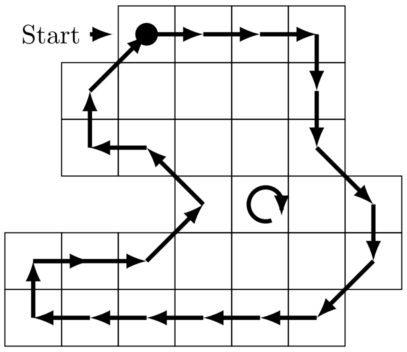


Fig. 5 A 2D object $\mathcal{B}$. Chain codes of $\mathcal{B}$ are $F8\,(\mathcal{B})$ = 0002212334445600754467 and $AF8\,(\mathcal{B})$ = *baacahbbbaaaaccahghcb*.

2.1 A helical chain code from a 3D object

We now describe the steps to obtain a helical chain code and a point cloud associated with a 3D object.

2.1.1 Voxelization

Given a 3D object in a polyhedral representation, the first step is to voxelize it. The voxelization can be done with the Binvox software [22, 23].

2.1.2 A helical chain code

The general idea is to decompose the voxelized 3D object into 2D slices. Then, we encode every 2D slice with the $AF8$ chain code. However, we need to encode every slice in a certain order to generate the helical path, which is crucial for reconstructing the 3D object.

A three-dimensional connected object $\mathcal{D}$ can be seen as a union of $n \geq 1$ z-slices: $\mathcal{D} = \bigcup_{i=1}^{n} \mathcal{D}_i$. Each z-slice $\mathcal{D}_i$ can be considered a 2D object, which may be disconnected. Every single $\mathcal{D}_i$ is the union of $m_i \in \mathbb{Z}_{\geq 0}$ connected components which are pairwise disjoint: $\mathcal{D}_i = \bigcup_{j=1}^{m_i} \mathcal{D}_{ij}$. Note that each connected component $\mathcal{D}_{ij}$ can also be considered as a 2D object.

Remark 2.2. As a consequence of the previous paragraph, we obtain that any 3D object $\mathcal{D}$ can be seen as a union of 2D objects $\mathcal{D}_{ij}$ that are pairwise disjoint:

$$\mathcal{D} = \bigcup_{i=1}^{n} \mathcal{D}_i = \bigcup_{i=1}^{n} \bigcup_{j=1}^{m_i} \mathcal{D}_{ij}.$$

The helical chain code of the 3D object $\mathcal{D}$ is obtained by the concatenation of the chain codes $AF8\left(\mathcal{D}_{ij}\right)$ and a few more symbols that help to encode an appropriate order.

Assume that the connected components $\mathcal{D}_{i1}, \mathcal{D}_{i2}, \ldots, \mathcal{D}_{im_i}$ in every slice $\mathcal{D}_i$ have been ordered from the closest to the furthest respect to the origin of coordinates. Let C_{ij} be the Cartesian coordinates of the center of the voxel that is the closest to the origin among all the voxels in the connected component $\mathcal{D}_{ij}$. Let ℓ_1 be the natural number such that $\mathcal{D}_{11}$ is adjacent to $\mathcal{D}_{21}$, $\mathcal{D}_{21}$ is adjacent to $\mathcal{D}_{31}, \ldots, \mathcal{D}_{(\ell_1-1)1}$ is adjacent to $\mathcal{D}_{\ell_1 1}$, and $\mathcal{D}_{\ell_1 1}$ is not adjacent to any another connected component D_{ij}. Define the concatenation

$$HCC^1\left(\mathcal{D}\right) = C_{11}AF8\left(\mathcal{D}_{11}\right) \cdots C_{\ell_1 1}AF8\left(\mathcal{D}_{\ell_1 1}\right).$$

Observe that we start with $\mathcal{D}_{11}$, which is the 2D component closest to the origin, and then we encode all the $\mathcal{D}_{ij}$'s that are adjacent to D_{11}. Now, we remove $\mathcal{D}_{11}, \ldots \mathcal{D}_{\ell_1 1}$ and repeat the previous step to find the chains of the regions: $HCC^2\left(\mathcal{D}\right), \ldots, HCC^t\left(\mathcal{D}\right)$, for a non-zero integer t.

Definition 2.3. Let $\mathcal{D}$ be a 3D object. The *helical chain code* of $\mathcal{D}$, denoted by $HCC\left(\mathcal{D}\right)$, is defined by

$$HCC\left(\mathcal{D}\right) = HCC^1\left(\mathcal{D}\right) y HCC^2\left(\mathcal{D}\right) y \ldots y HCC^t\left(\mathcal{D}\right),$$

where y is a symbol that helps to distinguish the chains $HCC^i\left(\mathcal{D}\right)$'s. The set of voxels encoded by a chain code $HCC^i\left(\mathcal{D}\right)$ is called a *region* of $\mathcal{D}$, and is denoted by $\mathcal{D}^i$.

Example 2.4. Fig. 6 (a) shows a 3D object $\mathcal{D}$. Observe that $\mathcal{D}$ is the union of nine 2D sets. Specifically,

$$\mathcal{D} = \mathcal{D}_{11} \cup \mathcal{D}_{21} \cup \mathcal{D}_{22} \cup \mathcal{D}_{31} \cup \mathcal{D}_{32} \cup \mathcal{D}_{41} \cup \mathcal{D}_{42} \cup \mathcal{D}_{51} \cup \mathcal{D}_{61}.$$

In order to compute the chain code $HCC(\mathcal{D})$, we find $AF8\left(\mathcal{D}_{11}\right)$. As $\mathcal{D}_{21}$ is adjacent to $\mathcal{D}_{11}$, we compute $AF8\left(\mathcal{D}_{21}\right)$. As $\mathcal{D}_{31}$ is adjacent to $\mathcal{D}_{21}$, we compute $AF8\left(\mathcal{D}_{31}\right)$. As $\mathcal{D}_{41}$ is adjacent to $\mathcal{D}_{31}$, we compute $AF8\left(\mathcal{D}_{41}\right)$. Finally, as $\mathcal{D}_{41}$ is not adjacent to the connected components on the slice $\mathcal{D}_5$, we compute $HCC^1\left(\mathcal{D}\right) = C_{11}AF8\left(\mathcal{D}_{11}\right) \cdots C_{41}AF8\left(\mathcal{D}_{41}\right)$, where C_{ij} represents the Cartesian coordinates of the center of the voxel in $\mathcal{D}_{ij}$ that is closest to the origin of the space. Observe that $HCC^1\left(\mathcal{D}\right)$ represents the chain code of the object in Fig. 6 (b), which is a region of Fig. 6 (a).

The next step is to remove the connected components $\mathcal{D}_{11}, \mathcal{D}_{21}, \mathcal{D}_{31}$ and $\mathcal{D}_{41}$ from the list of unvisited connected components of $\mathcal{D}$. Thus, we obtain Fig. 6 (c). As there are no more connected components in slice $\mathcal{D}_1$, we move to the next slice. We start with $\mathcal{D}_{22}$ and compute $AF8\left(\mathcal{D}_{22}\right)$. As $\mathcal{D}_{32}$ is adjacent to $\mathcal{D}_{22}$, we compute $AF8\left(\mathcal{D}_{32}\right)$. As $\mathcal{D}_{42}$ is adjacent to $\mathcal{D}_{32}$, we compute $AF8\left(\mathcal{D}_{42}\right)$. As $\mathcal{D}_{51}$ is adjacent to $\mathcal{D}_{42}$, we compute $AF8\left(\mathcal{D}_{51}\right)$. As $\mathcal{D}_{61}$ is adjacent to $\mathcal{D}_{51}$, we compute $AF8\left(\mathcal{D}_{61}\right)$. As there is no slice $\mathcal{D}_7$ in the object, we finally compute $HCC^2\left(\mathcal{D}\right)$ as the concatenation of the chains

$$C_{22}AF8\left(\mathcal{D}_{22}\right) C_{32}AF8\left(\mathcal{D}_{32}\right) C_{42}AF8\left(\mathcal{D}_{42}\right) \qquad \text{and} \qquad C_{51}AF8\left(\mathcal{D}_{51}\right) C_{61}AF8\left(\mathcal{D}_{61}\right),$$

where C_{ij} represents the Cartesian coordinates of the center of the voxel in $\mathcal{D}_{ij}$ that is closest to the origin of the space. Observe that $HCC^2(\mathcal{D})$ represents the chain code of the object in Fig. 6 (c), which is another region of Fig. 6 (a). The next step is to remove the connected components $\mathcal{D}_{22}, \mathcal{D}_{32}, \mathcal{D}_{42}, \mathcal{D}_{51}$ and $\mathcal{D}_{61}$ of $\mathcal{D}$ from the list of unvisited connected components. As there are no more connected components left to visit, we compute

$$HCC(\mathcal{D}) = HCC^1(\mathcal{D})\, y HCC^2(\mathcal{D})\,.$$

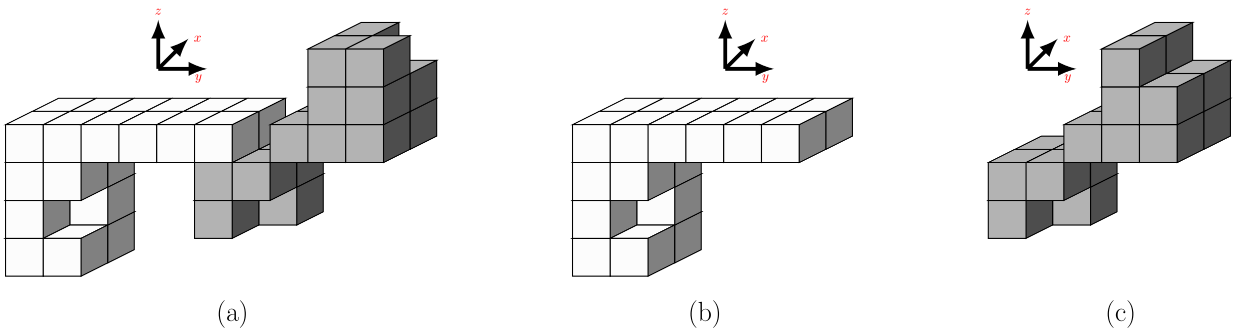


Fig. 6 (a) A 3D object $\mathcal{D}$. (b) One region of $\mathcal{D}$. (c) Another region of $\mathcal{D}$. The helical chain code $HCC(\mathcal{D})$ depends of $HCC_1(\mathcal{D})$, the chain code of (b), and $HCC_2(\mathcal{D})$, the chain code of (c).

Example 2.5. Fig. 7 shows a 3D object $\mathcal{D}$ composed by three different regions. The helical chain code $HCC(\mathcal{D})$ of the 3D object is composed by $HCC^1(\mathcal{D})$, $HCC^2(\mathcal{D})$, and $HCC^3(\mathcal{D})$, which are the chain codes of the regions. Specifically,

$$HCC(\mathcal{D}) = HCC^1(\mathcal{D})\, y HCC^2(\mathcal{D})\, y HCC^3(\mathcal{D})\,.$$

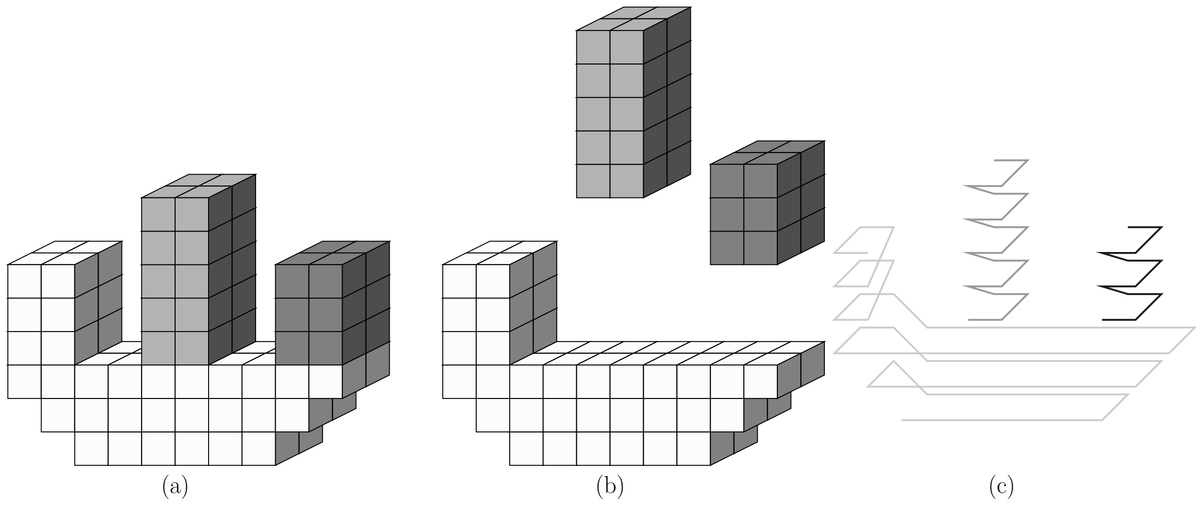


Fig. 7 (a) A 3D object $\mathcal{D}$. (b) The 3D object is composed of three regions. (c) The helical chain code $HCC(\mathcal{D})$ is composed by $HCC^1(\mathcal{D})$, $HCC^2(\mathcal{D})$, and $HCC^3(\mathcal{D})$, which are the chain codes of the regions.

Remark 2.6. Limitations of the chain code approach As we explained in Subsection 2.1.1, given a 3D object in a polyhedral representation, the first step is to voxelize the object. The voxelization, which can be done with the Binvox software [22, 23], captures the overall shape of the object, and it will result in practically imperceptible changes in the shape of the object as long as there is sufficient resolution. Thus, any chain code, helical or not, will keep all the information of the voxelized object. In other words, the chain code will preserve the whole topology of the original 3D object as long as the number of voxels used for the voxelization is sufficient.

We conclude that one of the limitations of the chain codes, helical or not, is given by the resolution of the voxelization. The voxelization is not the primary topic of this manuscript, but we want to mention that it can be tuned depending on the 3D objects being considered. For more information, see [22, 23].

Remark 2.7. Considerations about the helical chain code. The helical chain code has already been considered in the literature to encode voxelized objects; see [5] and the references therein. The helical chain code is crucial for this paper because, by Definition 2.3, it allows us to keep order on the voxels so the polyhedral reconstruction can be made.

As we explained in Remark 2.6, the helical chain codes preserve the whole information of the voxelization of the 3D object. In addition, among the several chain codes in the literature, the helical code is based on the A^* algorithm when moving from one slice to another. The A^* helps to find the closest voxel in the next slice, which is crucial for Algorithm 1 to join adjacent slices.

2.1.3 Low-density point clouds

Here, we obtain the low-density point clouds associated with the 3D object with the help of the helical chain code. The idea is based on [24], where the authors show digital straight segments of the contour shape of a 2D object can be obtained by identifying a particular string of symbols on the angle Freeman chain code. Then, after we identify the digital straight segments of the object in terms of the chain code, the point cloud is defined by the endpoints of the straight segments.

Proposition 2.8. [24] *A digital straight segment* (DSS) *of the contour shape of the* 2*D object* $\mathcal{B}$ *is given by the a substring in* $AF8\left(\mathcal{B}\right)$ *of the following form:*

$$Xa^p\left(Ya^q\right)^r,$$

where $X \in \{a,b,c,d,e,f,g,h\}, Y \in \{bh, hb\}$, *and* p,q,r *are non-negative integers that represent the number of times that a symbol, or set of symbols in case of the parenthesis, is concatenated.*

Every symbol in the chain code $AF8\left(\mathcal{B}\right)$ represents the center of a pixel that is in the contour shape of $\mathcal{B}$ [21].

Let $Xa^p\left(Ya^q\right)^r$ be a digital straight segment in the 2D object $\mathcal{B}$. The center of the pixel represented by X is called a *key point* of $\mathcal{B}$. By [24], it is clear that we can see the chain code $AF8\left(\mathcal{B}\right)$ as a concatenation of digital straight segments:

$$AF8\left(\mathcal{B}\right) = X_1a^{p_1}\left(Y_1a^{q_1}\right)^{r_1}\cdots X_\ell a^{p_\ell}\left(Y_\ell a^{q_\ell}\right)^{r_\ell}.$$

In this case, the set of key points of $\mathcal{B}$ is the set formed by the centers of the pixels defined by the symbols $X_1,\ldots,X_\ell$.

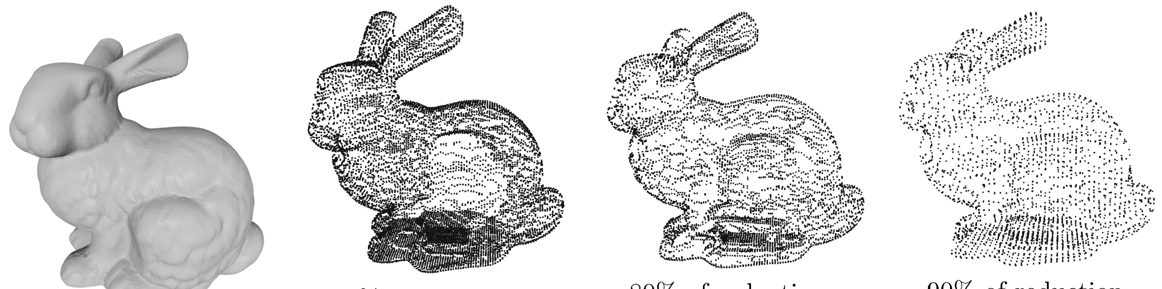
Definition 2.9. Let p,q,r, be non-negative integers, δ a positive integer, $\mathcal{D}$ a 3D object, and $\mathcal{D}_1,\ldots,\mathcal{D}_n$ the z-slices of $\mathcal{D}$. The low-density point cloud associated with p,q,r and δ that represents $\mathcal{D}$ is given by the set of key points of the components of $\mathcal{D}_1,\mathcal{D}_{1+\delta},\mathcal{D}_{1+2\delta},\ldots,\mathcal{D}_{1+i\delta},\ldots$.

Remark 2.10. Relation between the grammar and the point clouds. By Proposition 2.8, and as it is explained in detail in [8, 24], repetitions of the symbol a in the $AF8(\mathcal{B})$ string will indicate that there are a consecutive voxels in the 3D object are making contact by faces. Such voxels will produce a DSS. The combination of the symbol a with hb or bh in the $AF8(\mathcal{B})$ string will indicate that there are voxels in the 3D object that are forming a stair, which can approach the concept of DSS. Thus, the p, q, r, and δ parameters indicate the length of substrings we are looking for inside the 3D object. For example, if p is ten, we are looking for strings of ten consecutive face-connected voxels in the 3D object. But, if p is two, we are looking for strings of two consecutive face-connected voxels in the 3D object. Of course, there will be more strings of size two. Thus, there will be more DSS and more key points, resulting in a more dense point cloud.

The parameter δ indicates the slices where we are looking for DSS. For example, if δ is ten, we are looking for DSS every ten slices. If δ is *two*, we are looking for DSS every two slices in the 3D object.

As a summary, the density of the point cloud is inversely proportional to the p, q, r, and δ values. In other words, the larger the parameters p, q, r, and δ, the more details of the original object can be lost. In our experiments, we adjusted these values to achieve the densities shown in this article.

Example 2.11. Fig. 8 shows a 3D object and three associated point clouds for different p, q, r, and δ parameters to achieve distinct densities.



Original bunny. 65% of reduction. $p = 10$ $q = 10$ $r = 10$ $\delta = 4$. 80% of reduction. $p = 100$ $q = 100$ $r = 100$ $\delta = 5$. 90% of reduction. $p = 13$ $q = 13$ $r = 13$ $\delta = 11$.

Fig. 8 An original object and three different point clouds with distinct densities.

3 Polyhedral reconstruction

In this section, we describe the process of reconstructing a 3D object using the helical chain code from Subsection 2.1.2 and the point cloud from Subsection 2.1.3. We continue with the same notation from previous sections. In particular, from the definitions of z-slices, connected components and regions given above, a 3D object $\mathcal{D}$ can be seen as the following unions:

$$\mathcal{D} = \underbrace{\bigcup_{i=1}^{n} \mathcal{D}_i}_{z\text{-slices}} = \underbrace{\bigcup_{i=1}^{n} \bigcup_{j=1}^{m_i} \mathcal{D}_{ij}}_{\substack{\text{connected}\\ \text{components}}} = \underbrace{\bigcup_{i=1}^{t} \mathcal{D}^i}_{\text{regions}} . \tag{1}$$

Notation 3.1. As a point cloud of a 3D object $\mathcal{D}$ is given, we assume the integers p, q, r, and δ are fixed for this section. We use the following notation.

- $\mathcal{P}$ denotes the polyhedron $\mathcal{P}(\mathcal{D}, p, q, r, \delta)$ we want to reconstruct.
- $\mathbf{P}$ denotes the set of vertices of the polyhedron $\mathcal{P}(\mathcal{D}, p, q, r, \delta)$, which is given by the point cloud.
- $\mathbf{P}_i$ denotes the set of vertices that belong to the z-slice $\mathcal{D}_i$.
- $\mathbf{P}_{ij}$ denotes the set of vertices that belong to the connected component $\mathcal{D}_{ij}$.
- $\mathbf{P}^i$ denotes the set of vertices that belong to the region $\mathcal{D}^i$.

Observe that $\mathbf{P}_i \cap \mathbf{P}_j = \emptyset$, for $i \neq j$. A similar situation happens with the key points of the connected components and the regions. Thus, Eqs. (1) imply

$$\mathbf{P} = \bigcup_{i=1}^{n} \mathbf{P}_i = \bigcup_{i=1}^{n} \bigcup_{j=1}^{m_i} \mathbf{P}_{ij} = \bigcup_{i=1}^{t} \mathbf{P}^i .$$

In order to define the faces of the polyhedron $\mathcal{P}$, we need to determine the edges that belong to the faces. From now on, we focus on defining the edges rather than the faces. The set of all the edges that define the faces of the polyhedron $\mathcal{P}$ is denoted by $\mathcal{E}$.

Let v_1 and v_2 be two vertices in $\mathbf{P}$. The notation $\|v_1, v_2\|$ represents the Euclidean distance between the centers of the voxels v_1 and v_2. We now define a distance between voxels that belong to the same connected component.

Definition 3.2. Let v and v' be two vertices that belong to the same connected component $\mathcal{D}_{ij} = \{v_1, \dots v_\ell\}$. There exist two integers k and k' such that $v = v_k$ and $v' = v_{k'}$. Assume that $k < k'$. The *voxel distance* given by $\mathcal{D}$ is denoted and defined by

$$\|v_1, v_2\|_{\mathcal{D}} = \min \left\{ \|v_k, v_{k+1}\| + \cdots + \|v_{k'-1}, v_{k'}\|, \|v_{k'}, v_{k'+1}\| + \cdots + \|v_{k-1}, v_k\| \right\} .$$

It is not difficult to verify that $\|v_1, v_2\|_{\mathcal{D}}$ is a *distance function.* Indeed, it is straightforward to check that given any three vertices v, v' and v'' in the same connected component $\mathcal{D}_{ij}$, we have that

- $\|v, v\|_{\mathcal{D}} = 0$,
- $\|v, v'\|_{\mathcal{D}} = \|v', v\|_{\mathcal{D}}$, and
- $\|v, v'\|_{\mathcal{D}} \leq \|v, v''\|_{\mathcal{D}} + \|v'', v'\|_{\mathcal{D}}$.

Remark 3.3. If two vertices v_1 and v_2 are not on the same connected component, we define $\|v_1, v_2\|_{\mathcal{D}} := \|v_1, v_2\|$.

Definition 3.4. The *distance* between a voxel v and a set of voxels V_1 is denoted and defined by

$$\|v, V_1\| = \min_{v_1 \in V_1} \{\|v, v_1\|\}.$$

For any two key points v_1 and v_2 in $\mathbf{P}$, the notation $[v_1, v_2]$ represents the edge from the vertex v_1 to the vertex v_2.

We come to one of the main results of this section. Algorithm 1 details how to find all the edges $\mathcal{E}$ of the polyhedron $\mathcal{P}$.

Algorithm 1: Defines the edges, and as a consequence, the faces of $\mathcal{P}$

```
**Result:** $\mathcal{E}$, the set of edges of $\mathcal{P}$
$\mathcal{E} = \emptyset$;
for every connected component $\mathcal{D}_{ij}$ in $\mathcal{D}$ do
    Assume $\mathbf{P}_{ij} = \{v_1, \dots, v_\ell\}$ . We add the edges of every pair of consecutive vertices
      $[v_1, v_2], \dots, [v_{\ell-1}, v_\ell], [v_\ell, v_1]$ to $\mathcal{E}$;
    $\mathcal{C}_1 := \mathcal{D}_{(i+\delta)j}$ (adjacent connected component of $\mathcal{D}_{ij}$ in the same region);
    Let $\mathcal{C}_2, \dots, \mathcal{C}_\ell$ be the connected components of $\mathcal{D}_{i+\delta}$ such that each of them is either the first
      or last $z$-slice of another region $\mathcal{D}_{j'}$;
    Split $\mathcal{D}_{ij}$ in terms of the $\mathcal{C}_i$'s;
    for $1 \leq k \leq \ell$ do
        $\mathcal{D}_{ij}(k) = \{v \in \mathcal{D}_{ij} : \|v, \mathcal{C}_k\| < \|v, \mathcal{C}_{k'}\|, \text{ for } k \neq k'\}$
    end
    Take $\mathcal{C}'_1 := \mathcal{D}_{ij}(1), \dots, \mathcal{C}'_\ell := \mathcal{D}_{ij}(\ell)$;
    Let $\mathcal{C}'_{\ell+1}, \dots, \mathcal{C}'_{\ell'}$ be the connected components of $\mathcal{D}_i$ such that each of them is either the first
      or last $z$-slice of another region $\mathcal{D}_{j'}$;
    for $1 \leq k \leq \ell$ do
        $\mathcal{A}_k = \{v \in \mathcal{C}_k : \|v, \mathcal{D}_{ij}(k)\| < \|v, \mathcal{C}'_{k'}\|, \text{ for } k \neq k'\}$;
        We now join $\mathcal{D}_{ij}(k)$ and $\mathcal{A}_k$;
        $V := \mathbf{P} \cap \mathcal{D}_{ij}(k) = \{v_1, \dots, v_\ell\}, V' := \mathbf{P} \cap \mathcal{A}_k = \{v'_1, \dots, v'_{\ell'}\}$;
        The order in $V$ and $V'$ is defined by the helical chain code $HCC(\mathcal{D})$ , and $v_1$ and $v'_1$ have
          the property that $\|v_1, v'_1\|$ is minimum over all possible pairs $v, v'$;
        $n_1 := \|v'_1, v'_2\|_\mathcal{D} + \cdots + \|v'_{\ell-1}, v'_{\ell'}\|_\mathcal{D} + \|v'_{\ell'}, v'_1\|_\mathcal{D}$;
        $n_2 := \|v_1, v_2\|_\mathcal{D} + \cdots + \|v_{\ell-1}, v_\ell\|_\mathcal{D} + \|v_\ell, v_1\|_\mathcal{D}$;
        for every $v \in V$ do
            Take $v' \in V'$ such that the absolute value abs $\left(\|v', v'_1\|_\mathcal{D} - \frac{n_1}{n_2}\|v, v_1\|_\mathcal{D}\right)$ is minimum
              over all $v' \in V'$;
            We add the edge $[v, v']$ to $\mathcal{E}$
        end
        for every $v' \in V'$ do
            Take $v \in V$ such that the absolute value abs $\left(\|v', v'_1\|_\mathcal{D} - \frac{n_1}{n_2}\|v, v_1\|_\mathcal{D}\right)$ is minimum
              over all $v \in V$;
            We add the edge $[v', v]$ to $\mathcal{E}$
        end
        We eliminate the intersection of edges. See Fig. 9;
        $(V, V', \mathcal{E})$:=**Algorithm 2**$(V, V', \mathcal{E})$;
        We complete triangles. See Fig. 10;
        $(V, V', \mathcal{E})$:=**Algorithm 3**$(V, V', \mathcal{E})$;
    end
end
```

Remark 3.5. Even when Algorithm 1 depends on the metric $\|\cdot,\cdot\|_{\mathcal{D}}$, it can handle cases where the geometry of the object is highly non-linear or irregular; see Brain, Bunny, Cow, Dragon, and Heptoroid (a more complicated 3D object) in Fig. 12. One of the reasons is that Algorithm 1 handles every region separately, generating thus a single mesh for each connected region. Inside every region, Algorithm 1 utilizes the chain code to order and join the voxels of two consecutive slices. Thus, Algorithm 1 will always return a mesh between two consecutive slices and then between regions in a finite number of steps. We now give a more precise description.

One of the inputs of Algorithm 1 is the helical chain code of the 3D object. Such a code orders the surface voxels and decomposes the 3D object into different regions. The ordering allows us to uniquely define n_1 and n_2 between two adjacent slices, where each vertex of these adjacent slices starts from the nearest point. This characteristic enables us to determine how to create triangles either by connecting two points from the current slice with one point from the previous slice (n_1) or by connecting two points from the last slice with one from the current slice (n_2). Similarly, we can generate triangles between the current slice and the adjacent next slice.

As we explained in Remark 2.6, one of the main limitations of our process comes from the voxelization of the object. But even if the voxelization is large enough (in terms of the number of voxels), Algorithm 1 may fail with a low-density point cloud, for instance, when the p, q, r, and δ parameters are large (see Remark 2.10). In this scenario, the number of voxels between adjacent slices may differ considerably. Thus, an inaccurate surface reconstruction can come from a single point on a slice, disproportionately pulling all points from the previous one.

Algorithm 2: Eliminate intersections. See Fig. 9.

```
Data: $V, V', \mathcal{E}$
Result: $V, V', \mathcal{E}$, where there is no intersections between the edges from $V$ to $V'$
for every $v_k \in V$ and $v'_\ell \in V'$ such that $[v_k, v'_\ell] \in \mathcal{E}$ do
    if $[v_{k-1}, v'_{\ell+1}] \in \mathcal{E}$ then
        $d_1 := \|v_{k-1}, v_k\|_{\mathcal{D}}$;
        $d_2 := \|v'_\ell, v'_{\ell+1}\|_{\mathcal{D}}$;
        if $d_1 \leq d_2$ then
            Replace $[v_{k-1}, v'_{\ell+1}]$ by $[v_k, v'_{\ell+1}]$ in $\mathcal{E}$
        else
            Replace $[v_{k-1}, v'_{\ell+1}]$ by $[v_{k-1}, v'_\ell]$ in $\mathcal{E}$
        end
    end
    if $[v_{k+1}, v'_{\ell-1}] \in \mathcal{E}$ then
        $d_1 := \|v_{k+1}, v_k\|_{\mathcal{D}}$;
        $d_2 := \|v'_\ell, v'_{\ell-1}\|_{\mathcal{D}}$;
        if $d_1 \leq d_2$ then
            Replace $[v_{k+1}, v'_{\ell-1}]$ by $[v_k, v'_{\ell-1}]$ in $\mathcal{E}$
        else
            Replace $[v_{k+1}, v'_{\ell-1}]$ by $[v_{k+1}, v'_\ell]$ in $\mathcal{E}$
        end
    end
end
```

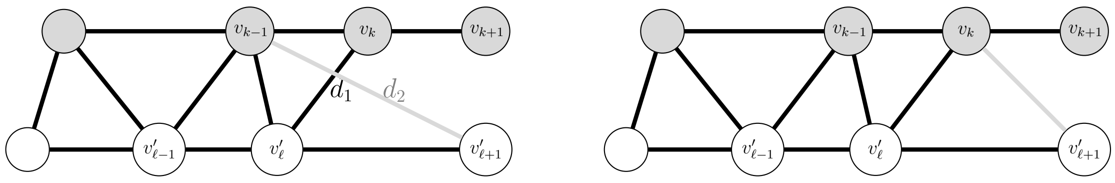


Fig. 9 Algorithm 2 eliminates intersections. The elimination depends on the distances d_1 and d_2.

Remark 3.6. In Algorithm 1, before calling Algorithm 2, the process has successfully joined every two slices. However, it may happen that extra joints with intersections were created (see Fig. 9). The goal of Algorithm 2 is to eliminate such intersections. We now give more details.

As we explained in Remark 3.5, Algorithm 1 highly depends on the helical chain code of the 3D object. Such a code orders the surface voxels and, consequently, the vertex from each slice. The ordering allows us to uniquely define d_1 and d_2 between two adjacent slices, where each vertex of these adjacent slices starts from a nearby point. Thus, Algorithm 2 creates triangles with no intersections either by connecting two points from the current slice with one point from the previous slice (d_1) or by connecting two points from the last slice with one from the current slice (d_2).

The limitation of Algorithm 2 arises when adjacent slices show significant differences in their vertex distribution. The triangulation process may produce irregular connections if one slice has widely spaced vertices and the next has a more compact distribution. Specifically, a single vertex from the less dense slice may disproportionately pull all the vertices from the denser slice, resulting in distorted triangles. Similarly, all the vertices of the smaller slice may be drawn toward a single vertex in another slice. This issue can arise when the value of δ is too large.

Algorithm 3: Complete triangles. See Fig. 10.

```
Data: $V, V', \mathcal{E}$
Result: $V, V', \mathcal{E}$, where the edges between $V$ and $V'$ form triangles
for every $v_k \in V$ and $v'_\ell \in V'$ such that $[v_k, v'_\ell] \in \mathcal{E}$ do
    if $[v_{k-1}, v'_{\ell-1}] \in \mathcal{E}, [v_{k-1}, v'_\ell] \notin \mathcal{E}, [v_k, v'_{\ell-1}] \notin \mathcal{E}$ then
        $d_1 := \|v_{k-1}, v'_\ell\|_{\mathcal{D}}$;
        $d_2 := \|v'_{\ell-1}, v_k\|_{\mathcal{D}}$;
        if $d_1 \leq d_2$ then
            Add $[v_{k-1}, v'_\ell]$ to $\mathcal{E}$
        else
            Add $[v_k, v'_{\ell-1}]$ to $\mathcal{E}$
        end
    end
    if $[v_{k+1}, v'_{\ell+1}] \in \mathcal{E}, [v_k, v'_{\ell+1}] \notin \mathcal{E}, [v_{k+1}, v'_\ell] \notin \mathcal{E}$ then
        $d_1 := \|v_k, v'_{\ell+1}\|_{\mathcal{D}}$;
        $d_2 := \|v'_\ell, v_{k+1}\|_{\mathcal{D}}$;
        if $d_1 \leq d_2$ then
            Add $[v_k, v'_{\ell+1}]$ to $\mathcal{E}$
        else
            Add $[v_{k+1}, v'_\ell]$ to $\mathcal{E}$
        end
    end
end
```

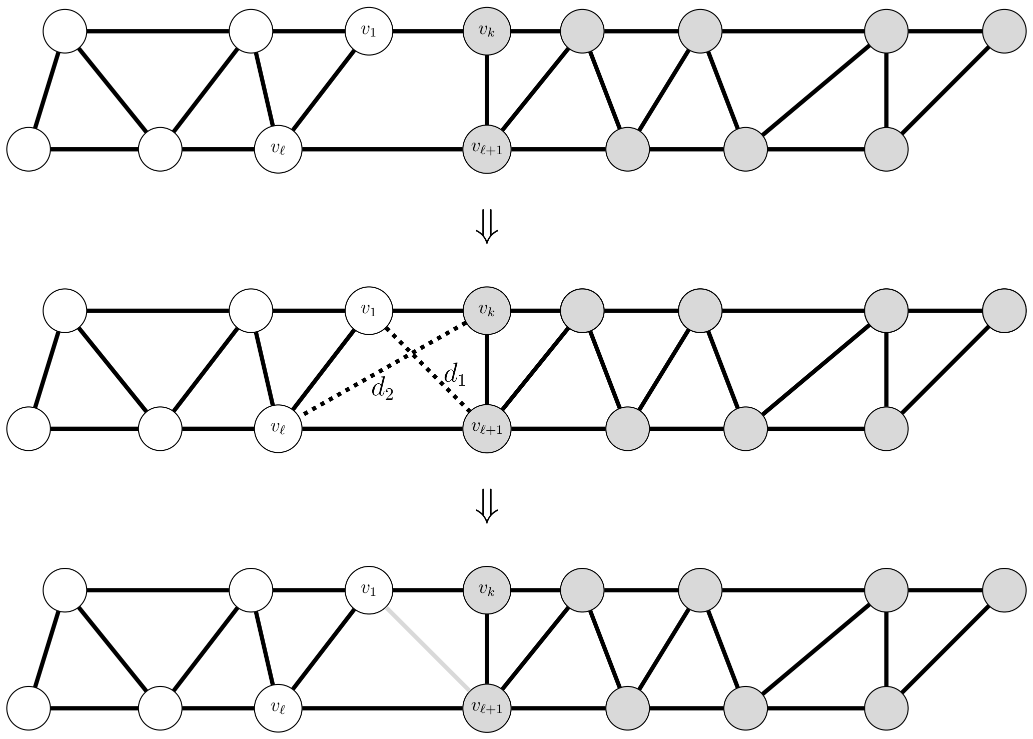


Fig. 10 Algorithm 3 creates triangles in the last slice of a region that is joined to two or more regions. The added triangle depends on the distances d_1 and d_2.

Remark 3.7. Independently of the shape of the object, by Algorithm 1, right before calling Algorithm 3 (after Algorithm 2), we have that at this stage, any two consecutive slices and regions are joined by edges with no intersections (by Algorithm 2). Sometimes, depending on the number and the shape of the regions, the vertices of the last slice of a region could be joined to two or more different regions. In this case, we need one more step, which is Algorithm 10, to create an appropriate triangulation. We now give a formal description.

Assume that $\mathcal{D}$ is a region that is joined to two different regions. Let $v_1, \ldots, v_k$ be the vertices of the last slice of $\mathcal{D}$. Assume that the vertices $v_1, \ldots, v_\ell$ are joined to one of the regions, and the vertices $v_{\ell+1}, \ldots, v_k$ are joined to another region. In this case, the vertices v_1, v_ℓ, $v_{\ell+1}$, and v_k will create a hole because there is no triangle; see Fig. 10. To address this situation, we propose Algorithm 3, which triangulates these four vertices within the same layer. Unlike Algorithms 1 and 2, which do not handle this scenario, Algorithm 3 aims to generate triangles with the shortest possible distance between d_1 and d_2, which is given by the helical chain code.

We present a summary of how Algorithm 1 works. We also show in Example 3.8 the main steps of Algorithm 1.

- Selects a connected component $\mathcal{D}_{ij}$.
- Adds the edges defined by two consecutive key points in $\mathcal{D}_{ij}$.
- Splits $\mathcal{D}_{ij}$ as a union $\bigcup \mathcal{D}_{ij}(k)$ and strategically selects subsets $\mathcal{A}_k$ of the connected components of the next slice $\mathcal{D}_{i+\delta}$.
- Creates edges between $\mathcal{D}_{ij}(k)$ and $\mathcal{A}_k$.
- Uses Algorithm 2 to eliminate intersections of edges between $\mathcal{D}_{ij}(k)$ and $\mathcal{A}_k$.
- Uses Algorithm 3 to complete triangles between $\mathcal{D}_{ij}(k)$ and $\mathcal{A}_k$.

It is important to remark that the previous steps are a superficial description of Algorithm 1. We remark that Algorithm 1 is strongly supported by the fact that the object has a helical chain code, which helps, for instance, to determine where the algorithm looks for subsets of the connected components to join them. Algorithm 1 also relies on the voxel distance and the relative distance, which helps to join two consecutive slices properly.

Algorithm 1 almost completes the polyhedron. A pair of cases need to be addressed: the *caps*, which are the first and last slices. For these cases, we use the very well-known Delaunay triangulation [25].

The Example 3.8 illustrates the steps of Algorithm 1.

Example 3.8. Fig. 11 shows the main steps of Algorithm 1. For simplicity, we are showing the view of the yz-plane. The four different colors of the object represent the four regions. (a) All the key points of the object. (b) The key points are delineated by the regions. (c) The algorithm generates the polyhedron for each region. (d) The algorithm strategically partitions the key points from the slices i and $i + \delta$. (e) The selected key points from slice i are connected to the selected key points from slice $i + \delta$. (f) The algorithm terminates when the polyhedron is acquired.

4 Results

Fig. 12 shows five original objects (from left to right and from top to bottom: Cow, Dragon, Brain, Bunny, and Heptoroid). In this section, we apply the polyhedral reconstruction from Algorithm 1 to different low-density point clouds representing these five original objects.

As we explained in Subsection 2.1.3, different values of p, q, r, and δ give rise to point clouds with distinct densities. Table 1 shows the (p, q, r, δ) parameters we used to achieve the point clouds with different percentages of simplification with respect to the original objects in Fig. 12. The point clouds associated with Fig. 12 and the (p, q, r, δ) parameters from Table 1 are shown from Fig. 13 (b) to Fig. 21 (b).

Finally, we apply Algorithm 1 to reconstruct the polyhedrons from the low-density point clouds and their respective helical chain codes. The results are shown from Fig. 13 (c) to Fig. 21 (c).

We can see from the experiments that our process, Algorithm 1, retrieves the main shape of the object from a low-density point cloud. However, we need to be careful because very few points may break the topology of the object, as the Cow with a 90% simplification shows in Fig. 14 (b).

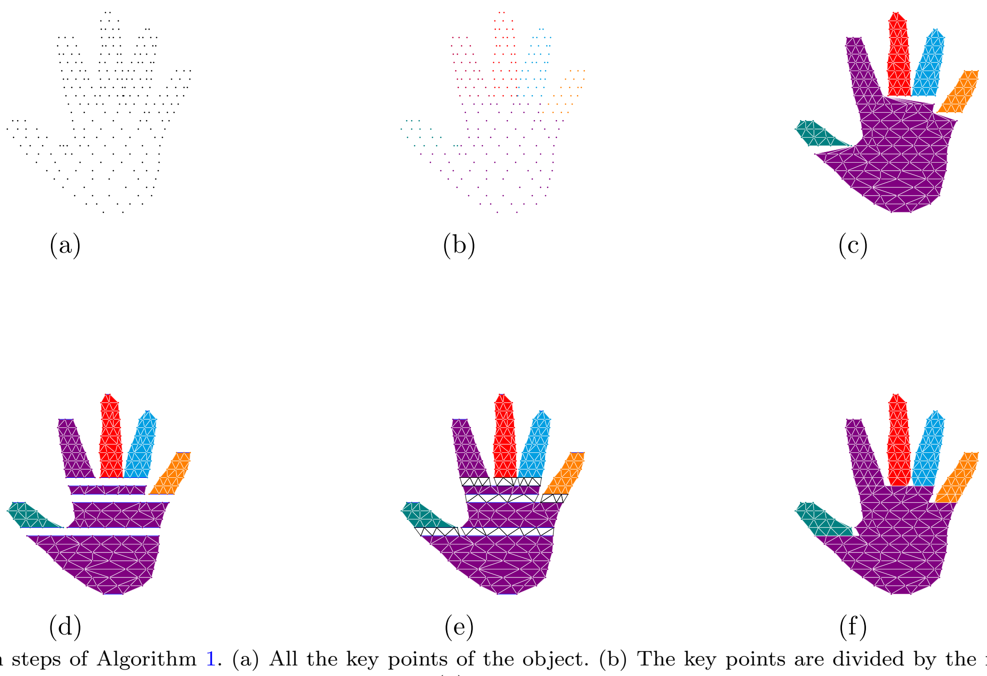


Fig. 11 Main steps of Algorithm 1. (a) All the key points of the object. (b) The key points are divided by the regions. (c) The algorithm creates the polyhedron for each region. (d) The algorithm strategically divides the key points from the slices i and $i + \delta$. (e) The selected key points from slice i are joined with the selected key points from slice $i + \delta$. (f) The algorithm finishes when the polyhedron is obtained.

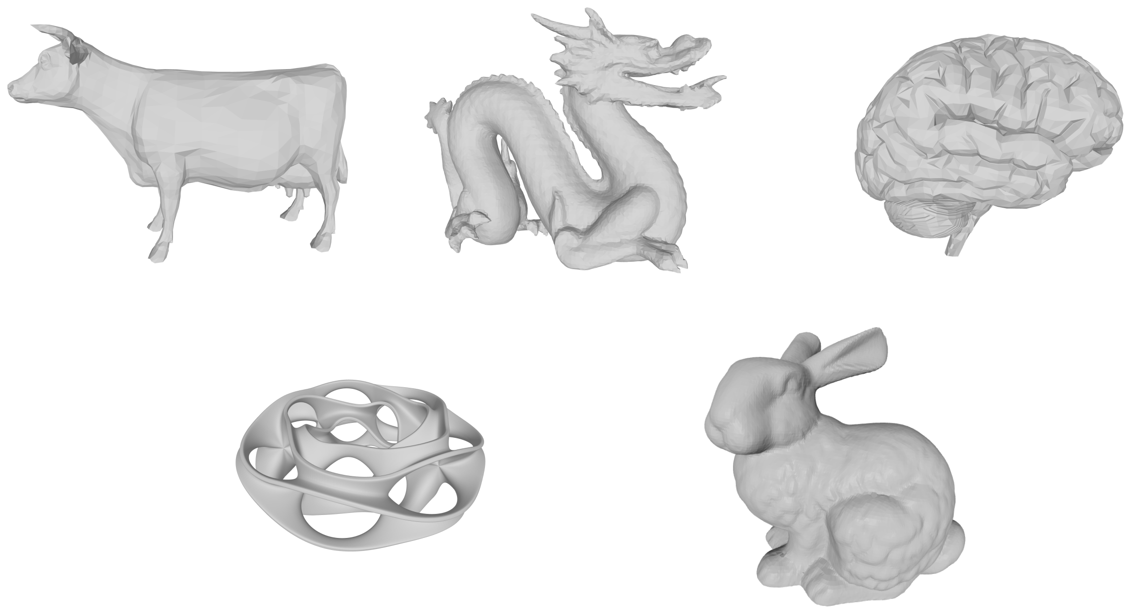


Fig. 12 Original set of three-dimensional objects.

4.1 Comparisons

Here, we compare our method, Algorithm 1, with other approaches in the literature to reconstruct polyhedrons from point clouds. We start by defining two metrics between sets of points.

Object	# Points (original)	(p, q, r, δ)	Simplification	# Points (simplified)
Cow	2,903	(100,100,100,17)	65%	1,007
Cow	2,903	(100,100,100,50)	90%	290
Dragon	22,998	(18,18,18,5)	65%	8,017
Brain	18,844	(100,100,100,17)	65%	6,488
Heptoroid	286,678	(2,3,1,2)	65%	100,337
Bunny	35,947	(10,10,10,4)	65%	12,406
Bunny	35,947	(9,10,8,5)	75%	8,978
Bunny	35,947	(100,100,100,5)	80%	7,308
Bunny	35,947	(13,13,13,11)	90%	3,591

Table 1 Different (p, q, r, δ) parameters to achieve distinct simplification percentages.

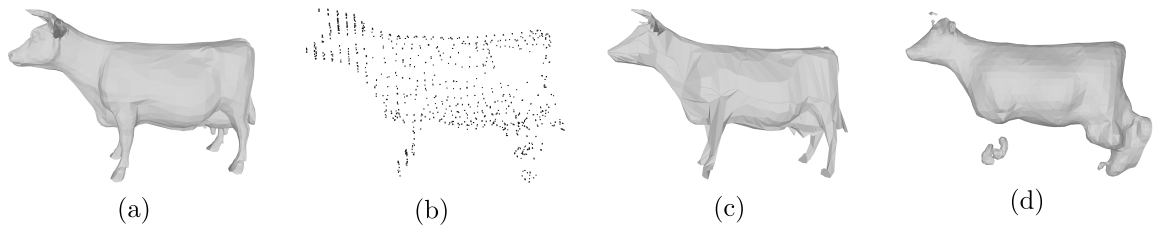


Fig. 13 (a) Original Cow. (b) Point cloud with a 65% reduction. (c) Reconstruction using our method, Algorithm 1. (d) Reconstruction using MeshLab [26].

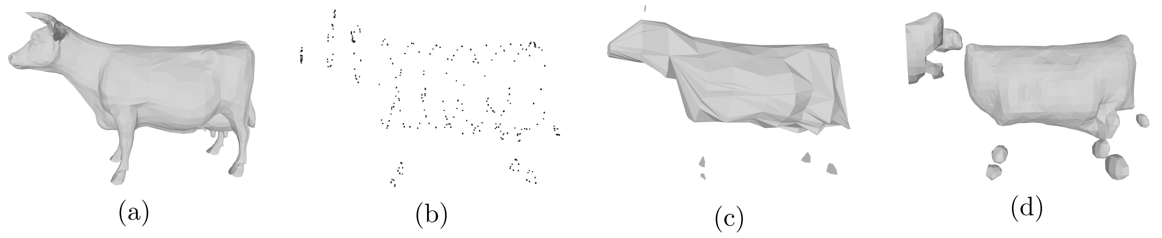


Fig. 14 (a) Original Cow. (b) Point cloud with a 90% reduction. (c) Reconstruction using our method, Algorithm 1. (d) Reconstruction using MeshLab [26].

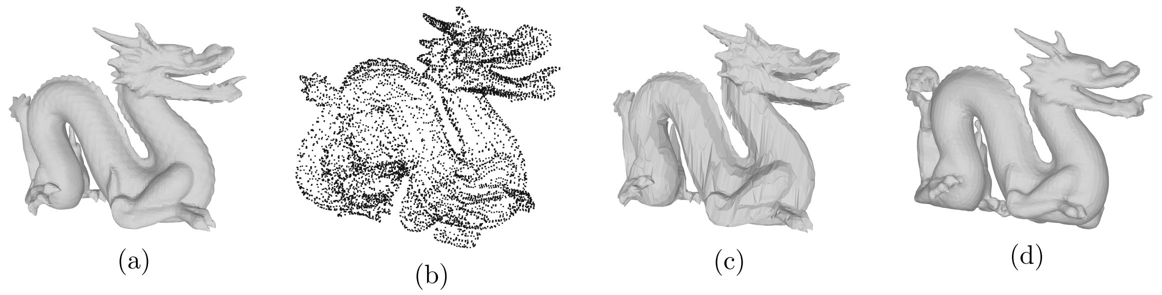


Fig. 15 (a) Original Dragon. (b) Point cloud with a 90% reduction. (c) Reconstruction using our method, Algorithm 1. (d) Reconstruction using MeshLab [26].

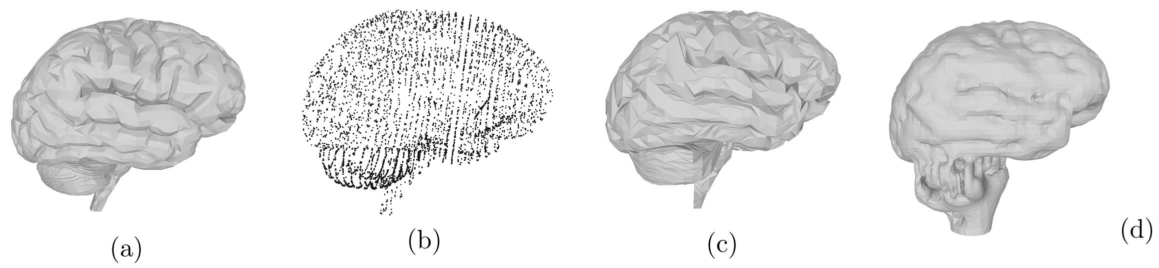


Fig. 16 (a) Original Brain. (b) Point cloud with a 65% reduction. (c) Reconstruction using our method, Algorithm 1. (d) Reconstruction using MeshLab [26].

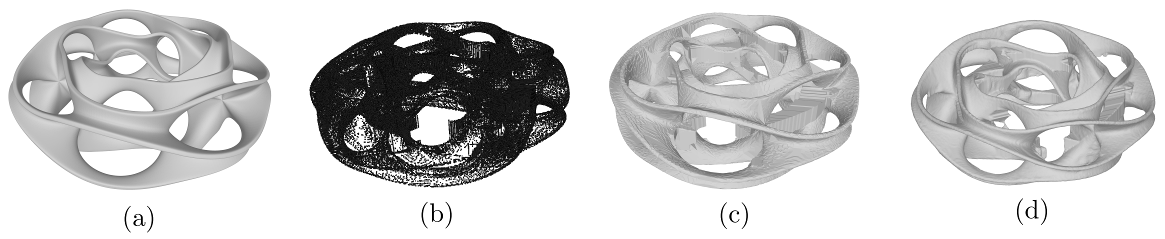


Fig. 17 (a) Original Heptorid. (b) Point cloud with a 65% reduction. (c) Reconstruction using our method, Algorithm 1. (d) Reconstruction using MeshLab [26].

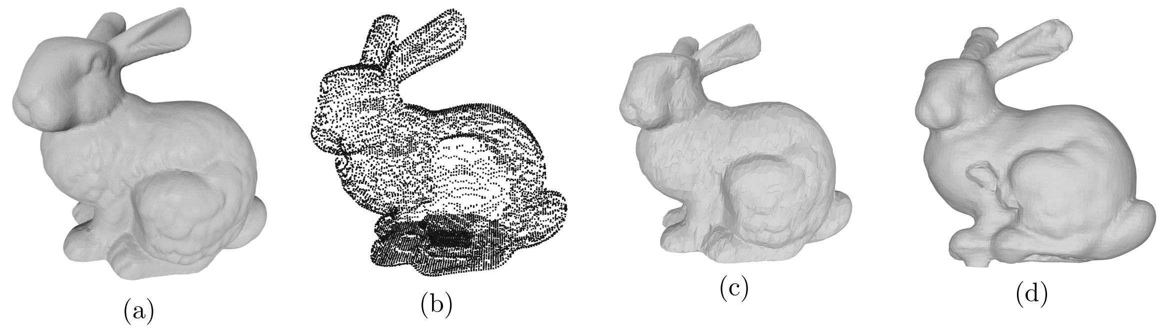


Fig. 18 (a) Original Bunny. (b) Point cloud with a 65% reduction. (c) Reconstruction using our method, Algorithm 1. (d) Reconstruction using MeshLab [26].

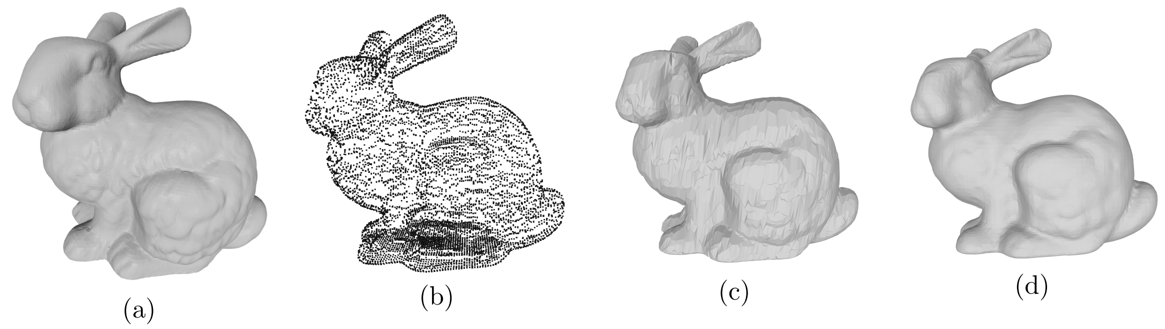


Fig. 19 (a) Original Bunny. (b) Point cloud with a 75% reduction. (c) Reconstruction using our method, Algorithm 1. (d) Reconstruction using MeshLab [26].

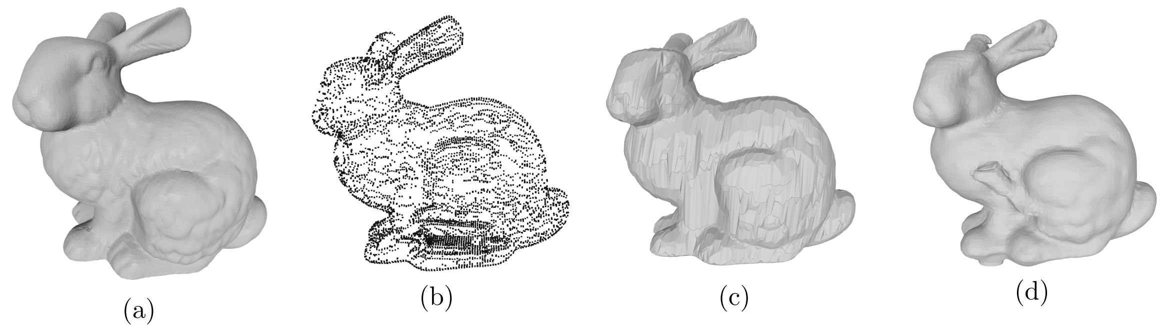


Fig. 20 (a) Original Bunny. (b) Point cloud with an 80% reduction. (c) Reconstruction using our method, Algorithm 1. (d) Reconstruction using MeshLab [26].

Given two finite sets of points $A := \{a_1, \ldots, a_p\}$ and $B := \{b_1, \ldots, b_q\}$, the **Hausdorff distance** [27] between A and B is denoted and defined as

$$H(A, B) = \max(h(A, B), h(B, A)),$$

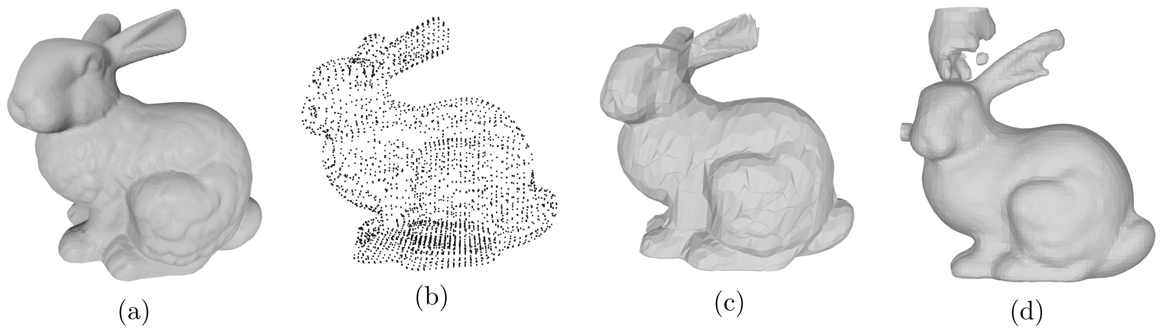


Fig. 21 (a) Original Bunny. (b) Point cloud with a 90% reduction. (c) Reconstruction using our method, Algorithm 1. (d) Reconstruction using MeshLab [26].

where

$$h(A, B) = \max_{a \in A} \min_{b \in B} \|a - b\|$$

and $\|\cdot\|$ denotes the Euclidean norm on the points of A and B. More information about the Hausdorff distance can be found in [27].

Given two finite sets of points $A = \{a_1, \dots, a_p\}$ and $B = \{b_1, \dots, b_q\}$, the **Chamfer distance** [27] between A and B is denoted and defined as

$$d_{\text{Chamfer}}(A, B) = \frac{1}{|A|} \sum_{a \in A} \min_{b \in B} \|a - b\|^2 + \frac{1}{|B|} \sum_{b \in B} \min_{a \in A} \|b - a\|^2,$$

where $\|\cdot\|$ is the Euclidean norm and $|A|$ and $|B|$ are the cardinalities of A and B, respectively.

We compare our method, Algorithm 1, with other approaches in the literature to reconstruct polyhedrons from point clouds. The reconstruction using MeshLab [26] is shown from Fig. 13 (d) to Fig. 21 (d). Visually, we can see that MeshLab [26] tries to make the object softer, which can make the reconstruction lose some characteristics with respect to the original 3D object. Another advantage of Algorithm 1 is that it can recover the shape even when there are fewer points. Tables 2 and 3 show the comparisons, using the Hausdorff and Chamfer distances, respectively, between the original 3D object and the polyhedron obtained with our method, Algorithm 1, and the polyhedral reconstruction using MeshLab [26].

Hausdorff Distance									
Polyhedral reconstruction	Cow 65%	Cow 90%	Dragon 65%	Brain 65%	Heptoroid 65%	Bunny 65%	Bunny 75%	Bunny 80%	Bunny 90%
Algorithm 1	**0.807**	**1.279**	**0.010**	**0.562**	0.876	**1.355**	1.470	1.592	**1.715**
MeshLab [26]	1.015	1.341	0.011	1.102	**0.873**	1.491	**1.424**	**1.584**	3.917

Table 2 A comparative analysis of polyhedral reconstruction using the Hausdorff distance between the original 3D object and the polyhedron obtained with our method, Algorithm 1, and the polyhedral reconstruction using MeshLab [26]. The lowest values are marked in bold.

Chamfer Distance									
Polyhedral reconstruction	Cow 65%	Cow 90%	Dragon 65%	Brain 65%	Heptoroid 65%	Bunny 65%	Bunny 75%	Bunny 80%	Bunny 90%
Algorithm 1	**0.287**	**0.701**	0.006	**0.206**	**0.334**	**0.347**	0.283	0.312	**0.562**
MeshLab [26]	0.333	0.709	**0.005**	0.353	0.379	0.398	**0.260**	**0.308**	0.852

Table 3 A comparative analysis of polyhedral reconstruction using the Chamfer distance between the original 3D object and the polyhedron obtained with our method, Algorithm 1, and the polyhedral reconstruction using MeshLab [26]. The lowest values are marked in bold.

We compare our method, Algorithm 1, with [28] and [14]. Table 4 shows the results. As expected, Algorithm 1 runs faster when there is a simplification because it uses fewer points.

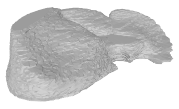
Model	Points	Triangular mesh number	Time of [14]	Time of [28]	Time of Algorithm 1
Dragon	41842	80118	37.91	38.51	35.28
Bunny	35337	67662	26.77	26.61	27.22
Bunny(after simplification)	8304	17455	11.53	/	4.33

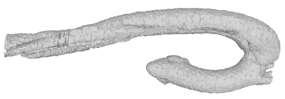
Table 4 Times for polyhedrons reconstruction.

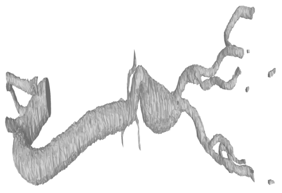
5 Medical applications

We present in this section the results of applying our method, Algorithm 1, to the reconstruction of 3D medical images that are represented or transmitted by slices.

Fig. 22 shows the polyhedrons obtained using our method, Algorithm 1, to different sets of slices. We give more details below.


(a) Polyhedral Liver


(b) Polyhedral Type-B Aortic Dissection


(c) Polyhedral Abdominal Aortic Aneurysm

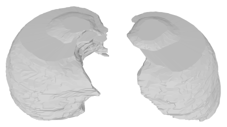

(d) Polyhedral Lungs

Fig. 22 Polyhedrons using our method, Algorithm 1, to different sets of slices.

5.1 Liver

Fig. 23 shows a sample of slices of a liver. The result of applying Algorithm 1 to the complete set of slices [29] is a polyhedron that represents the liver, which is shown in Fig 24.

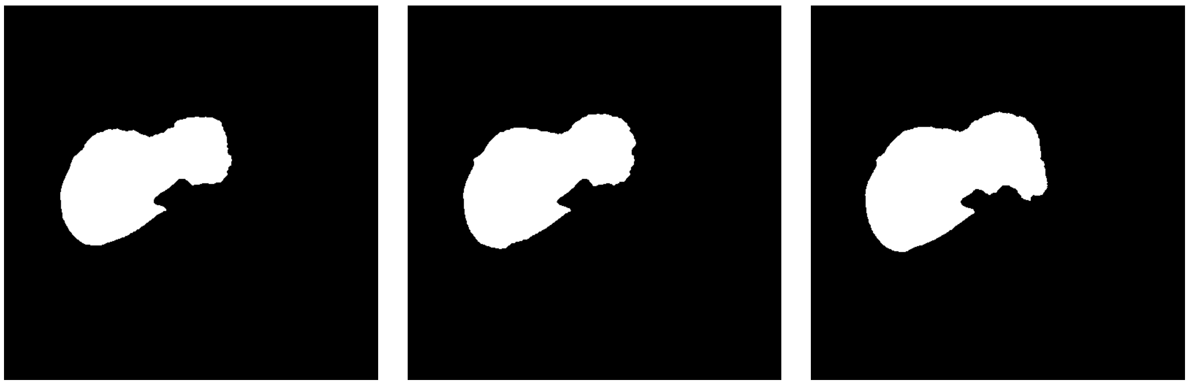


Fig. 23 A sample of slices of a liver. The full set consists of 94 slices of a liver [29].

5.2 Type-B Aortic Dissection

Fig. 25 shows a sample of slices of a Type-B Aortic Dissection. The result of applying our method, Algorithm 1, to the complete set of slices [30] is a polyhedron that represents the Type-B Aortic Dissection, which is shown in Fig 26.

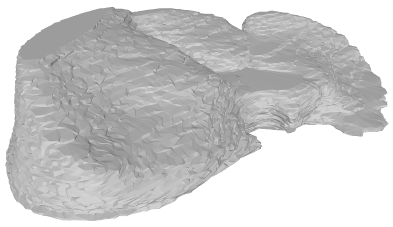


Fig. 24 A polyhedron that represents the liver obtained from the complete set of slices [29] and Algorithm 1.

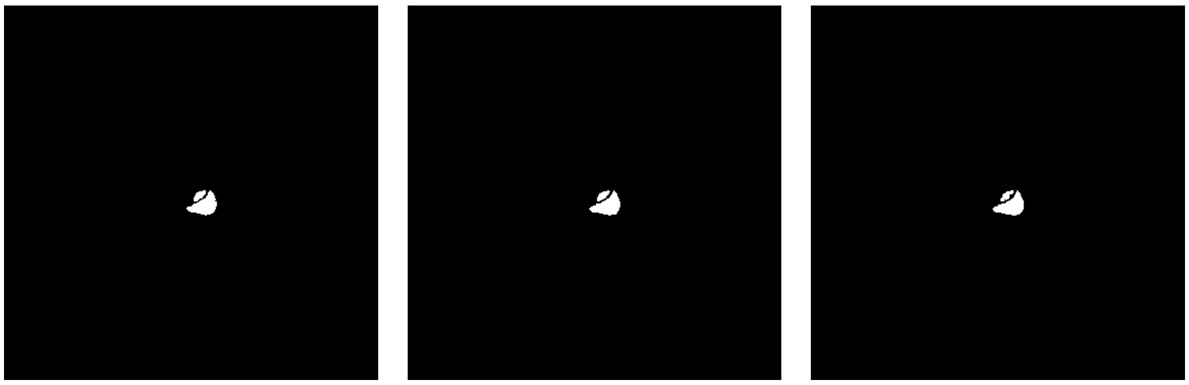


Fig. 25 A sample of slices of a Type-B Aortic Dissection. The full set consists of 316 slices of a Type-B Aortic Dissection [30].

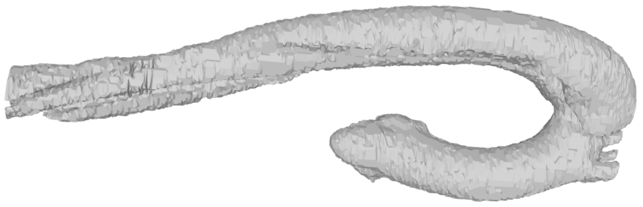


Fig. 26 A polyhedron that represents the Type-B Aortic Dissection obtained from the complete set of slices [30] and Algorithm 1.

5.3 Abdominal Aortic Aneurysm

Fig. 27 shows a sample of slices of a tree showing an abdominal aortic aneurysm. The result of applying Algorithm 1 to the complete set of slices [5] is a polyhedron that represents the tree showing an abdominal aortic aneurysm, which is shown in Fig 28.

5.4 Lung nodule

Fig. 29 shows a sample of slices of a lung nodule. The result of applying Algorithm 1 to the complete set of slices [31] is a polyhedron that represents the lung nodule, which is shown in Fig 30.

5.5 Comparisons

Chain codes, which are strings of symbols representing the contour of a 2D image or a 3D object, have been extensively considered in the literature because of their compressive properties. This compact and efficient representation allows us to manipulate and analyze the shape of the object easily. In addition, the string represents a data reduction that improves performance in applications that handle large volumes

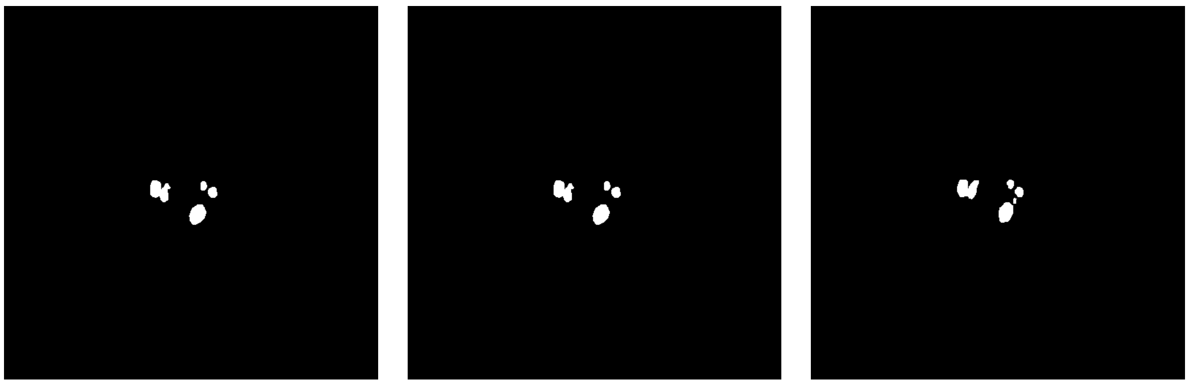


Fig. 27 A sample of slices of a tree showing an abdominal aortic aneurysm. The full set consists of 340 slices of a tree showing an abdominal aortic aneurysm [5].

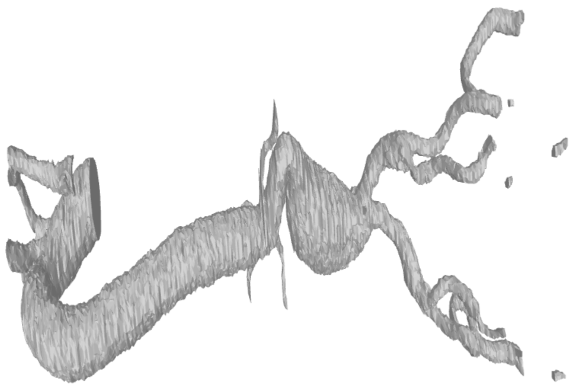


Fig. 28 A polyhedron that represents a tree showing an abdominal aortic aneurysm obtained from the complete set of slices [5] and Algorithm 1.

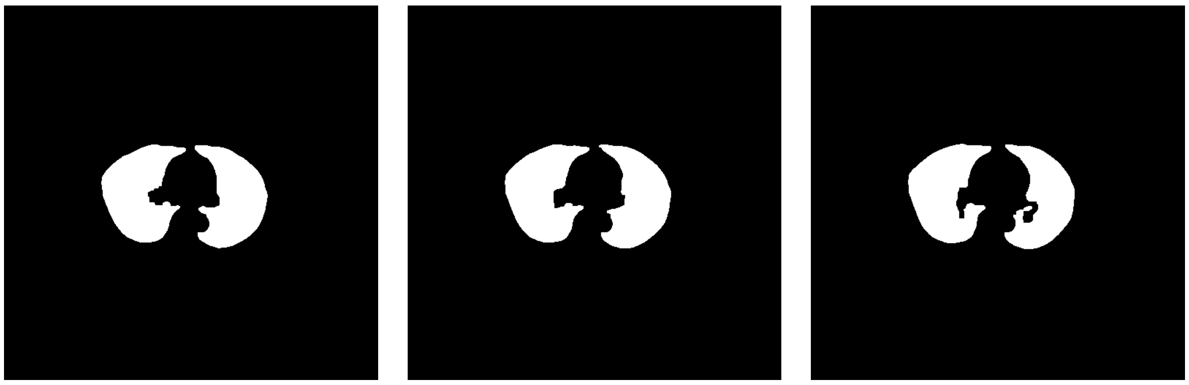


Fig. 29 A sample of slices of a lung nodule. The full set consists of 57 slices of the lung nodule [31].

of data. In [32], the authors describe a lossless data compression block-sorting algorithm that reorders the input text for simpler compression. The authors proved that the algorithm, which is based on the move-to-front coding, achieves speeds similar to Lempel-Ziv algorithms and compression rates comparable to top statistical models. In [33], the authors use interpolative coding for lossless chain code compression. The compression pipelines include the burrows-wheeler transform, move-to-front transform and enhanced interpolative coding.

The combination of a chain code with a compression algorithm provides an effective solution for compressing, improving storage efficiency and transmission speed [7, 21]. PAQ file archivers [32, 33], which are distributed as free software under the GNU General Public License (https://mattmahoney.net/dc/), are a family of lossless data compressors based on context mixing. Context mixing is an algorithm in

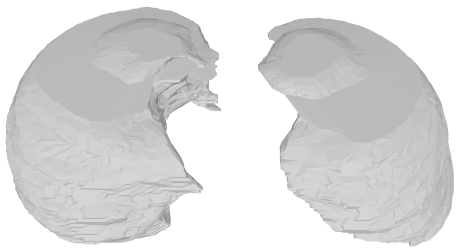


Fig. 30 A polyhedron that represents the lung nodule obtained from the complete set of slices [31] and Algorithm 1.

which predictions of the next symbol in the chain are calculated by combining two or more statistical models to produce a better prediction. In particular, among the PAQ archivers, one of them is PAQ8l, which we used for lossless compression in our data type. It is known for its high compression rate, although it may be slower than other archivers because of its focus on the highest possible compression.

In Table 5, we compare the size of the objects from Fig. 22 using different representations and PAQ8l. We see that the size of the polyhedron obtained from Algorithm 1 is less than the size of the object using the slices representation. Table 5 shows that the .PLY files obtained from our method, Algorithm 1, have better compression than the original files in a .PNG format. The high compression ratios are because we represented the surfaces of the objects in terms of triangulations (edges and vertices) in comparison with the volumetric data from the original 3D object.

3D medical dataset	Size Slices (png)	Size Polyhedron from Algorithm 1 (ply)
Liver [29]	67 KB	**30 KB**
Type-B Aortic Dissection [30]	358 KB	**34 KB**
Abdominal aortic aneurysm [5]	74 KB	**19 KB**
Lung nodule [31]	448 KB	**26 KB**

Table 5 Compression obtained when Paq8l [33] is applied to the complete set of slices and the polyhedron obtained with our method, Algorithm 1.

Remark 5.1. For medical applications, it is crucial to consider that our approach, Algorithm 1, will fail if there is no segmentation. In other words, if the set of slices contains different organs that are not segmented, then Algorithm 1 will fail.

6 Conclusions and further work

In this work, we presented an algorithm that constructs a polyhedron that reliably represents a 3D object from a low-density point cloud and a helical chain code. The proposed algorithm takes advantage of the compression that comes from the chain code. In addition, as the chain code contains the information on the contour shape of the 3D object, we use context-free grammar to detect the so-called key points of the 3D object and the digital straight segments. The experiments show that even with a 65% of simplification, which means only 35% of the points, the constructive algorithm reliably reconstructs the original 3D object. We showed that our proposed algorithm can reconstruct 3D medical images represented or transmitted by slices.

Algorithm 1 highly depends on the (p, q, r, δ) parameters and the helical chain code. As explained in Remark 2.10, there is a strong relation between the density of the points cloud and the (p, q, r, δ) parameters. Future work is to develop a formal statistical analysis to determine the (p, q, r, δ) for which a given 3D object starts to lose its main topological properties.

For medical applications, Algorithm 1 considers that the set of slices contains different organs that are already segmented. Future work is to modify Algorithm 1 so it works even if the organs are not segmented.

Algorithm 1 depends on the voxelization and the chain code of the original 3D object. Future work would be to adapt Algorithm 1 so it can work with 3D objects that are not voxelized. This could be a huge improvement with potential applications to virtual reality or 3D scene reconstruction.

Declarations

Availability of data and material

Data will be made available on request.

Competing interests

The authors have no conflicts of interest to declare that are relevant to the content of this article.

Funding

Hiram H. López was partially supported by the NSF grants DMS-2201094 and DMS-2401558. Hermilo Sánchez-Cruz was partially supported by Universidad Autónoma de Aguascalientes under grant PII24-4.

Authors' contributions

All the authors contributed equally to the development of this manuscript.

References

[1] Montaño-Serrano, V.M., Jacinto-Villegas, J.M., Vilchis-González, A.H., Portillo-Rodríguez, O.: Artificial vision algorithms for socially assistive robot applications: A review of the literature. Sensors **21**(17), 5728 (2021) https://doi.org/10.3390/s21175728

[2] Lai, J.W., Cheong, K.H.: Adoption of virtual and augmented reality for mathematics education: A scoping review. IEEE Access **10**, 13693–13703 (2022) https://doi.org/10.1109/ACCESS.2022.3145991

[3] McCloskey, K., Turlip, R., Ahmad, H.S., Ghenbot, Y.G., Chauhan, D., Yoon, J.W.: Virtual and augmented reality in spine surgery: A systematic review. World Neurosurgery **173**, 96–107 (2023) https://doi.org/10.1016/j.wneu.2023.02.068

[4] Han, X.-F., Laga, H., Bennamoun, M.: Image-based 3d object reconstruction: State-of-the-art and trends in the deep learning era. IEEE Transactions on Pattern Analysis and Machine Intelligence **43**(5), 1578–1604 (2021) https://doi.org/10.1109/TPAMI.2019.2954885

[5] Aldemir, E., Arturo Tapia Dueñas, O., Emre Kavur, A., Tohumoglu, G., Sánchez-Cruz, H., Alper Selver, M.: Chain code strategy for lossless storage and transfer of segmented binary medical data. Expert Systems with Applications **216**, 119449 (2023) https://doi.org/10.1016/j.eswa.2022.119449

[6] Pervan, B., Tomic, S., Ivandic, H., Knezovic, J.: Midom?a dicom-based medical image communication system. Applied Sciences **13**(10) (2023) https://doi.org/10.3390/app13106075

[7] Erdo?an Aldemir, G.T., Selver, M.A.: Binary medical image compression using the volumetric run-length approach. The Imaging Science Journal **67**(3), 123–135 (2019) https://doi.org/10.1080/13682199.2019.1565695

[8] Tapia-Dueñas, O.A., Sánchez-Cruz, H., López, H.H.: 3d object simplification using chain code-based point clouds. Multimedia Tools and Applications **82**(6), 9491–9515 (2023) https://doi.org/10.1007/s11042-022-13588-3

[9] Brüel-Gabrielsson, R., Ganapathi-Subramanian, V., Skraba, P.: Topology-aware surface reconstruction for point clouds. Computers Graphics Forum **39**(5), 197–207 (2020) https://doi.org/10.1111/cgf.14079

[10] Zhu, L., Kukko, A., Virtanen, J.-P., Hyyppä, J., Kaartinen, H., Hyyppä, H., Turppa, T.: Multisource point clouds, point simplification and surface reconstruction. Remote Sensing **11**(22) (2019) https://doi.org/10.3390/rs11222659

[11] Jarvis, B., Choi, G.P.T., Hockman, B., Morrell, B., Bandopadhyay, S., Lubey, D., Villa, J., Bhaskaran, S., Bayard, D., Nesnas, I.A.: 3d shape reconstruction of small bodies from sparse features. IEEE Robotics and Automation Letters **6**(4), 7089–7096 (2021) https://doi.org/10.1109/LRA.2021.3097273

[12] Chen, X., Ravikumar, N., Xia, Y., Attar, R., Diaz-Pinto, A., Piechnik, S.K., Neubauer, S., Petersen, S.E., Frangi, A.F.: Shape registration with learned deformations for 3d shape reconstruction from sparse and incomplete point clouds. Medical Image Analysis **74**, 102228 (2021) https://doi.org/10.1016/j.media.2021.102228

[13] Wang, P., Wang, Z., Xin, S., Gao, X., Wang, W., Tu, C.: Restricted delaunay triangulation for explicit surface reconstruction. ACM Transactions on Graphics **41**(5) (2022) https://doi.org/10.1145/3533768

[14] Li, H.-A., Zhang, M., Yu, K., Qi, X., Hua, Q., Zhu, Y.: R3mr: Region growing based 3d mesh reconstruction for big data platform. IEEE Access **8**, 91740–91750 (2020) https://doi.org/10.1109/ACCESS.2020.2993964

[15] Sheikhpour, R., Berahmand, K., Mohammadi, M., Khosravi, H.: Sparse feature selection using hypergraph laplacian-based semi-supervised discriminant analysis. Pattern Recognition **157**, 110882 (2025) https://doi.org/10.1016/j.patcog.2024.110882

[16] Sheikhpour, R., Mohammadi, M., Berahmand, K., Saberi-Movahed, F., Khosravi, H.: Robust semi-supervised multi-label feature selection based on shared subspace and manifold learning. Information Sciences **699**, 121800 (2025) https://doi.org/10.1016/j.ins.2024.121800

[17] Parvaiz, A., Khalid, M.A., Zafar, R., Ameer, H., Ali, M., Fraz, M.M.: Vision transformers in medical computer vision—a contemplative retrospection. Engineering Applications of Artificial Intelligence **122**, 106126 (2023) https://doi.org/10.1016/j.engappai.2023.106126

[18] Fischer, F., Selver, M.A., Gezer, S., Dicle, O., Hillen, W.: Systematic parameterization, storage, and representation of volumetric dicom data. Journal of Medical and Biological Engineering **35**(6), 709–723 (2015) https://doi.org/10.1007/s40846-015-0097-5

[19] Popescu, D., Marinescu, R., Laptoiu, D., Deac, G., Cotet, C.: Dicom 3d viewers, virtual reality or 3d printing – a pilot usability study for assessing the preference of orthopedic surgeons. Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine **235**(9), 1014–1024 (2021) https://doi.org/10.1177/09544119211020148

[20] Freeman, H.: On the encoding of arbitrary geometric configurations. IRE Transactions on Electronic Computers **EC-10**(2), 260–268 (1961) https://doi.org/10.1109/TEC.1961.5219197

[21] Liu, Y.K., Žalik, B.: An efficient chain code with huffman coding. Pattern Recognition **38**(4), 553–557 (2005) https://doi.org/10.1016/j.patcog.2004.08.017

[22] Min, P.: Binvox. `http://www.patrickmin.com/binvox` (2004 - 2019)

[23] Nooruddin, F.S., Turk, G.: Simplification and repair of polygonal models using volumetric techniques. IEEE Transactions on Visualization and Computer Graphics **9**(2), 191–205 (2003)

[24] Tapia-Dueñas, O.A., Sánchez-Cruz, H.: Context-free grammars to detect straight segments and a novel polygonal approximation method. Signal Processing: Image Communication **91**, 116080 (2021) https://doi.org/10.1016/j.image.2020.116080

[25] Delaunay, B.: Sur la sphere vide. Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk **7**(793-800), 1–2 (1934)

[26] Kazhdan, M., Bolitho, M., Hoppe, H.: Poisson surface reconstruction. In: Proceedings of the Fourth Eurographics Symposium on Geometry Processing, vol. 7 (2006)

[27] Huttenlocher, D.P., Klanderman, G.A., Rucklidge, W.J.: Comparing images using the hausdorff distance. IEEE Transactions on pattern analysis and machine intelligence **15**(9), 850–863 (1993)

[28] Nan, L., Wonka, P.: Polyfit: Polygonal surface reconstruction from point clouds. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 2372–2380 (2017). https://doi.org/10.1109/ICCV.2017.258

[29] Kavur, A.E., Selver, M.A., Dicle, O., Barlş, M., Gezer, N.S.: Chaos - combined (ct-mr) healthy abdominal organ segmentation challenge data (2019) https://doi.org/10.5281/zenodo.3362844

[30] Yao, Z., Xie, W., Zhang, J., Dong, Y., Qiu, H., Yuan, H., Jia, Q., Wang, T., Shi, Y., Zhuang, J., *et al.*: Imagetbad: A 3d computed tomography angiography image dataset for automatic segmentation of type-b aortic dissection. Frontiers in Physiology **12**, 732711 (2021)

[31] Zhou, X.: segmented images. figshare. Dataset. (2015). https://doi.org/10.6084/m9.figshare.1579435.v1

[32] Mahoney, M.V.: Adaptive weighing of context models for lossless data compression. (2005). https://api.semanticscholar.org/CorpusID:17386893

[33] Mahoney, M.: Data compression programs. Overview over PAQ based compression software. http://mattmahoney. net/dc . . . (2009)