Contribution to Structural Parameters Computation: Volume Models and Methods

by: Irving A. Cruz-Matías

Advisor: Dra. Dolors Ayala Vallespí

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy.
Ph.D. Program in Computing
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya (UPC)

Barcelona, November, 2013

A la memoria de mi querida abuela Catalina Salvador (†).

Agradecimientos (Acknowledgements)

La finalización de un gran proyecto como el desarrollo de esta tesis doctoral no hubiese sido posible sin el apoyo de algunas personas e instituciones. Espero que estas líneas sirvan para expresar mi más profundo y sincero agradecimiento a todos aquellos que con su ayuda han colaborado en la realización del presente trabajo.

En primer lugar quiero dar gracias a Dios por todas las bendiciones y retos puestos en el camino, por permitirme soñar y por hacer que muchos de esos sueños se volvieran realidad.

Quiero agradecer a mi tutora de tesis, Dra. Dolors Ayala por su paciencia y sobre todo por su valioso tiempo y conocimientos invertidos en el desarrollo de esta tesis. A mis profesores de la UPC de quienes obtuve valiosos conocimientos para mi formación profesional. A los profesores y compañeros de investigación del *Grup de Modelització i Visualització de dades Biomèdiques* por su apoyo y su amistad.

Un agradecimiento muy especial merece la comprensión, cariño y ánimo recibidos de toda mi familia, principalmente de mis padres Soledad y Humberto, y mi hermano Julio, cuyas palabras de apoyo a través de la distancia me han servido para culminar un logro más.

Agradezco tambíen a todas aquellas personas que me han brindado su amistad y que directa o indirectamente han contribuido al desarrollo de este trabajo, en especial a Cristina por su ayuda en la redacción del inglés.

Finalmente quiero agradecer a las instituciones que creyeron en mi financiando el presente trabajo de investigación. En primer lugar está la Agencia Española de Cooperación Internacional, por la beca de doctorado MAEC-AECID, y al Consejo Nacional de Ciencia y Tecnología (CONACYT-México) por la beca complementaria.

A todos ellos, muchas gracias.

Resumen

El Bio-Diseño Asistido por Computadora (Bio-CAD), y la experimentación *in-silico* están teniendo un creciente interés en aplicaciones biomédicas, en donde se utilizan datos científicos provenientes de muestras reales para calcular parámetros estructurales que permiten evaluar propiedades físicas. Las tecnologías de adquisición de imagen no invasivas como la TC, μTC o IRM, y el crecimiento constante de las prestaciones de las computadoras, permiten la adquisición, procesamiento y visualización de datos científicos con creciente grado de complejidad.

El cálculo de parámetros estructurales está basado en la existencia de dos fases (o espacios) en la muestra: la sólida, que puede corresponder al hueso o material, y la fase porosa o vacía, por tanto, tales muestras son representadas como volúmenes binarios. El modelo de representación más común para estos conjuntos de datos es el modelo de vóxeles, el cuál es una extensión natural a 3D de los mapas de bits 2D. En esta tesis se utilizan el modelo *Extreme Verrtices Model* (EVM) y un nuevo modelo propuesto, *the Compact Union of Disjoint Boxes* (CUDB), para representar los volúmenes binarios en una forma mucho más compacta. El modelo EVM almacena sólo un subconjunto ordenado de vértices de la frontera del objeto mientras que el modelo CUDB mantiene una lista compacta de cajas.

En esta tesis se proponen métodos para calcular los siguientes parámetros estructurales: distribución del tamaño de los poros, conectividad, orientación, esfericidad y redondez. La distribución del tamaño de los poros ayuda a interpretar las características de las muestras porosas permitiendo a los usuarios observar los rangos de diámetro más comunes de los poros mediante picos en un gráfica. La conectividad es una propiedad topológica relacionada con el género del espacio sólido, mide el nivel de interconectividad entre los elementos, y es un indicador de las características biomecánicas del hueso o de otros materiales. La orientación de un objeto puede ser definida por medio de ángulos de rotación alrededor de un conjunto de ejes ortogonales. La esfericidad es una medida de que tan esférica es una partícula , mientras que la redondez es la medida de la nitidez de sus aristas y esquinas.

En el estudio de estos parámetros se trabaja con muestras reales escaneadas a alta resolución que suelen generar conjuntos de datos enormes, los cuales requieren una gran cantidad de memoria y mucho tiempo de procesamiento para ser analizados. Por esta razón, se presenta un nuevo método para simplificar volúmenes binarios de una manera progresiva y sin pérdidas. Este método genera una secuencia de niveles de detalle de los objetos, en donde cada objeto es un volumen englobante de los objetos previos. Además de ser utilizado como apoyo en el cálculo de parámetros estructurales, este método puede ser de utilizado en otras tareas como transmisión progresiva, detección de colisiones y cálculo de volumen de interés.

Como parte de una investigación multidisciplinaria, se han desarrollado dos aplicaciones prácticas para calcular parámetros estructurales de muestras reales. Un software para la detección automática de puntos de viscosidad característicos en muestras de rocas de basalto y vidrios, y una aplicación para calcular la esfericidad y redondez de formas complejas en un conjunto de datos de sílice.

Abstract

Bio-CAD and *in-silico* experimentation are getting a growing interest in biomedical applications where scientific data coming from real samples are used to compute structural parameters that allow to evaluate physical properties. Non-invasive imaging acquisition technologies such as CT, μCT or MRI, plus the constant growth of computer capabilities, allow the acquisition, processing and visualization of scientific data with increasing degree of complexity.

Structural parameters computation is based on the existence of two phases (or spaces) in the sample: the solid, which may correspond to the bone or material, and the empty or porous phase and, therefore, they are represented as binary volumes. The most common representation model for these datasets is the voxel model, which is the natural extension to 3D of 2D bitmaps. In this thesis, the *Extreme Vertices Model* (EVM) and a new proposed model, the *Compact Union of Disjoint Boxes* (CUDB), are used to represent binary volumes in a much more compact way. EVM stores only a sorted subset of vertices of the object's boundary whereas CUDB keeps a compact list of boxes.

In this thesis, methods to compute the next structural parameters are proposed: pore-size distribution, connectivity, orientation, sphericity and roundness. The pore-size distribution helps to interpret the characteristics of porous samples by allowing users to observe most common pore diameter ranges as peaks in a graph. Connectivity is a topological property related to the genus of the solid space, measures the level of interconnectivity among elements, and is an indicator of the biomechanical characteristics of bone or other materials. The orientation of a shape can be defined by rotation angles around a set of orthogonal axes. Sphericity is a measure of how spherical is a particle, whereas roundness is the measure of the sharpness of a particle's edges and corners.

The study of these parameters requires dealing with real samples scanned at high resolution, which usually generate huge datasets that require a lot of memory and large processing time to analyze them. For this reason, a new method to simplify binary volumes in a progressive and lossless way is presented. This method generates a level-of-detail sequence of objects, where each object is a bounding volume of the previous objects. Besides being used as support in the structural parameter computation, this method can be practical for task such as progressive transmission, collision detection and volume of interest computation.

As part of multidisciplinary research, two practical applications have been developed to compute structural parameters of real samples. A software for automatic detection of characteristic viscosity points of basalt rocks and glasses samples, and another to compute sphericity and roundness of complex forms in a silica dataset.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations and Acronyms

General abbreviations and acronyms used in this thesis:

AABB	Axis-Aligned Bounding Box.
BD	Backward Difference.
BP-O	Bounding-Planes Octree.
B-Rep	Boundary Representation model.
BSP	Binary Space Partition.
CAD	Computer-Aided Design.
CC	Connected Component.
CCL	Connected-Component Labeling.
CSG	Constructive Solid Geometry.
CT	Computed Tomography.
CVP	Characteristic Viscosity Point.
ECDT	Extended Convex Differences Tree.
EV	Extreme Vertice.
EVM	Extreme Vertices Model.
FD	Forward Difference.
FEM	Finite Element Method.
KC	Krumbein's Chart.
LOD	Level of Detail.
LSGA	Line Skeleton Graph Analysis.
MIP	Mercury Intrusion Porosimetry.
MRI	Magnetic Resonance Imaging.
NMR	Nuclear Magnetic Resonance.
OBB	Oriented Bounding Box.
OPP	Orthogonal Pseudo-Polyhedra.
OUDB	Ordered Union of Disjoint Boxes.
PLA	Polylactic Acid.
PLY	Polygon File Format / Stanford Triangle Format.

SB	Semi-Boundary model.
TIFF	Tagged Image File Format.

Abbreviations and acronyms defined in this thesis:

ABN	A-Backward Neighbor.
AFN	A-Forward Neighbor.
BBN	B-Backward Neighbor.
BFN	B-Forward Neighbor.
BN	Backward Neighbor.
BOPP	Bounding Orthogonal Pseudo-Polyhedra.
CUDB	Compact Union of Disjoint Boxes.
PEVE	Progressive Extreme Vertices Encoding.

CHAPTER 1

Introduction

1.1 Motivation

Volume modeling and visualization is a widely studied field. There is a large number of contributions on general models and analysis and visualization methods. Nevertheless, the continuous increase of applications that use this technology, as well as the size and complexity of data that we have to deal with, have defined new challenges.

New imaging acquisition technologies, such as micro/nano computed tomography (CT), magnetic resonance imaging (MRI) or nuclear magnetic resonance (NMR), plus the constant growth of computer capabilities, allow the acquisition, processing and visualization of scientific data from several fields such as biomaterials, medicine, mineralogy, geology, etc, and with increasing degree of complexity. These acquisition methods produce volume datasets in a non-invasive way, which can be used to compute physical properties.

The development of the bioengineering field has yielded to the appearance of a new application of *Computer-Aided Design*, *Bio-CAD*. It refers to the computation of structural parameters from images of samples and it is widely used in many applications. Structural parameters allow us to measure physical properties of a sample such as osteoporosis degree of bones, suitability of biomaterials to be used as implants, transport and distribution of fluids into rocks for petrographic purposes, grain shape and sphericity index in silica sand for industrial and manufacturing applications, among others.

Structural parameters can be of different typology. All of them are based on the existence of two or more phases (or spaces) in the sample. The most common cases involve two phases, the solid, which may correspond to the bone or material, and the empty or porous phase and, therefore, they are represented as binary volume datasets. Some of the structural parameters are simply the volume of different phases or the contact surface between them, but others such as connectivity are more complex. Other parameters consist in morphological information of these phases, and in this case, an appropriate representation model must be defined. Many of the methods for structural parameters computation use the voxel model, while others use alternative representation models such as octrees or kd-trees.

Porosisity and the pore-size distribution are experimentally evaluated with a device called porosimeter, which introduces mercury into a sample at increasing pressures, causing fluid to flow through smaller apertures, and then the corresponding intruded volumes are obtained. This experimental method has been simulated using volume datasets. There are other methods to compute the porosity. Heuristic methods that cover the pore space with overlapping spheres into the cavities and granulometry methods that perform morphological openings using geometric elements. However, virtual porosimetry and heuristic methods rely on a previous computation of a 1D skeleton (medial line or curve skeleton) or 2D skeleton (medial surface), which are very time-consuming processes.

Connectivity, a topological property related to the genus of the solid space, measures the level of interconnectivity among elements. It can be obtained from the Euler characteristic, using the voxel model. In the solid modeling field it can also be computed from a polyhedron or from a triangulated surface. The performance variability of existing methods is mainly caused by the number of basic geometric elements to analyze (voxels, triangles, vertices, etc.).

This thesis aims to provide new methods to compute structural parameters, which improve the performance of existing methods. Consequently, we work with representation models that allow us to analyze geometrically the porous and solid spaces in an efficient and compact way. We work with existing models and also propose a new suitable model that allows the information to be compacted and speeds up the methods.

Besides, the study of these parameters requires dealing with the large size and complexity of the models that result from the capturing devices, which affects the computation speed up of their characteristics, disk storage and rendering efficiency. Simplification techniques can diminish these problems. Moreover, in some situations it is advantageous to exchange an exact geometric representation of an object for an approximated one, which can retain the enough relevant information and be processed more efficiently. Therefore, we also devise a new method to simplify binary volumes. As the proposed method is progressive, lossless, and produces bounding objects, it can be practical for visualization, progressive transmission, and collision detection, among other tasks. For example, simplified models can be used for collision detection in prosthesis placement simulation, where, using simplified bounding forms, we can discard certain zones and then work directly with the original model.

1.2 Objectives

The main objective of this thesis is to contribute to structural parameters computation with the development of new methods. In order to accomplish this, a set of specific objectives have been established:

- Devise a new representation model for binary volumes that allows the information to be compacted and provides fast neighborhood operations.

- Provide a new approach for virtual porosimetry, which avoids the skeleton computation, and gives a better performance compared with existing skeleton-based approaches.

- Provide a method to compute the connectivity of binary volumes based on the proposed decomposition model, that gives a better performance compared with methods that use alternative models.
- Provide geometric methods to evaluate the sphericity and roundness indices of objects, whose results have a high correlation with existing indices used in geology.
- Devise a framework to simplify binary volume datasets and general orthogonal pseudo-polyhedra, in a progressive and lossless way, with good compression ratio, approximation quality and run time. Moreover, apply this simplification framework to the computation of the mentioned structural parameters.
- Work with datasets coming from real samples and take part in multidisciplinary projects with other research teams.
- Develop two practical applications: an automatic detection of characteristic viscosity points of basalt rocks and glasses samples, and sphericity and roundness computation of complex forms in silica datasets.
- Incorporate all the developed work into a software platform to offer the aforementioned models and methods.

1.3 Thesis Outline

Besides this chapter, the thesis is organized as follows:

Chapter 2: Background.

In this chapter, the relationship between binary volume datasets and orthogonal pseudo-polyhedra (OPP) is explained. The main representation models used in this thesis, the Extreme Vertices Model (EVM) and the Ordered Union of Disjoint Boxes (OUDB), represent binary volumes as OPP. Therefore, for the purpose of this thesis, these terms are often used indistinctly.

This chapter also reviews alternative representation models and describes the state of the art related to structural parameters computation and simplification methods.

Chapter 3: An Improved Decomposition Model for OPP.

This chapter presents a new decomposition model for OPP: the Compact Union of Disjoint Boxes (CUDB). This model is an improved version of OUDB. CUDB has many desirable features versus OUDB, such as less storage size and a better efficiency in the connected-component labeling (CCL) process. Algorithms for conversion between CUDB and other models, and basic algorithms used in subsequent processes of structural parameters computation and simplification, such as CCL and exact collision (adjacency) detection, are presented. CUDB performance is experimentally analyzed with phantom and real datasets.

Chapter 4: Structural Parameters Computation.

This chapter presents several methods to compute structural parameters of samples using CUDB and EVM. The measured parameters are: pore-size distribution, connectivity, orientation, sphericity and roundness.

The first method simulates mercury intrusion porosimetry. This method does not require skeleton computation and is an iterative process that considers the diameters corresponding to pressures. At each iteration, geometric tests detect throats for the corresponding diameter and a CCL process collects the region invaded by the mercury. The method obtains the pore-size distribution of the porous sample.

The second method computes the Euler characteristic (χ) and the genus of a volume dataset. This method is derived from the classical method used with a voxel model and the computation of χ and the genus is achieved by analyzing the connectivity among CUDB-boxes and using a CCL process.

The third method computes the orientation, sphericity and roundness of objects. Orientation is defined by three mutually orthogonal unit vectors, sphericity is estimated based on volume and surface measures, and roundness is estimated with a ray-casting-like approach.

All the methods are tested both with phantom and real datasets and compared with previous methods based on the voxel model, and other alternative models.

Chapter 5: Lossless Orthogonal Simplification.

This chapter presents a new approach to simplify general OPP represented with EVM. The method is incremental and produces a level-of-detail sequence of OPP, where any object of this sequence bounds the previous objects. The sequence finishes with the axis-aligned bounding box. Simplification is achieved using a new strategy called *merging faces*, which relies on the application of 2D Boolean operations. A data structure to encode in a progressive and lossless way the generated sequence is also devised. This method is compared in storage size and approximation quality with other methods that also produce bounding objects. This simplification method is applied to the computation of the aforementioned structral parameters.

Chapter 6: Practical Applications.

In this chapter, two practical applications to compute structural parameters are presented.

The first practical application is a customized software analysis tool to automatically detect characteristic viscosity points (CVP) from a collection of 2D images of basalt rocks and glasses samples. This tool was developed to help determine the temperatures corresponding to CVP together with the Hot-stage microscopy technique.

The second practical application is a software to analyze silica nano datasets. Silica sand is normally required to have grains of an approximately uniform size. Sphericity and roundness, two of the more important physical properties of silica sand, are analyzed.

Chapter 7: Conclusions and Future Work.

This chapter summarizes the contributions, and sketches future work lines.

CHAPTER 2

Background

2.1 Introduction

This chapter describes the background and the state of the art of the main issues that are analyzed in this thesis. It describes the main representation models of volume data where the relationship between binary volume datasets and orthogonal pseudo-polyhedra (OPP) is explained. The Extreme Vertices Model (EVM) and the Ordered Union of Disjoint Boxes (OUDB), the most referenced representation models in this thesis are reviewed in more detail.

This chapter also reviews the structural parameters studied in this thesis and discusses the state of the art of simplification methods, mainly those methods that produce progressive approximations which are also bounding volumes.

2.2 Volume Models

Volume datasets coming from acquisition technologies must be modeled in some way before being processed or visualized in order to obtain the desired results, i.e., once a 3D model is obtained from a source, data needs to be represented in a generic model and then, fitted with a volume representation model. Three possible definitions of volume are [102]:

1. It can be viewed as the process of modeling volume data, as volume scanning devices produce a value of a dependent quantity at various locations in space.
2. It can be thought of as a generalization in dimension to surface modeling.
3. It can be viewed as the means to provide the input to the volume rendering.

Hence, volume modeling can be thought as starting with some 3D objects, manipulating them and resulting in new 3D objects [139]. The generated object model can be used further in different ways.

Nowadays, there are diverse proposed volume models to represent sampled data and each one of these provides distinct features and has different advantages and drawbacks. Therefore, the use of one or another mainly depends on what kind of operations we need to apply and which information we want to extract.

Volume data are usually registered as values in different locations of the 3D space, the sampling locations can be organized to form either structured or unstructured meshes [66], where structured meshes can be uniform, rectilinear or curvilinear. Figure 2.1 depicts samples of structured meshes and an unstructured mesh.

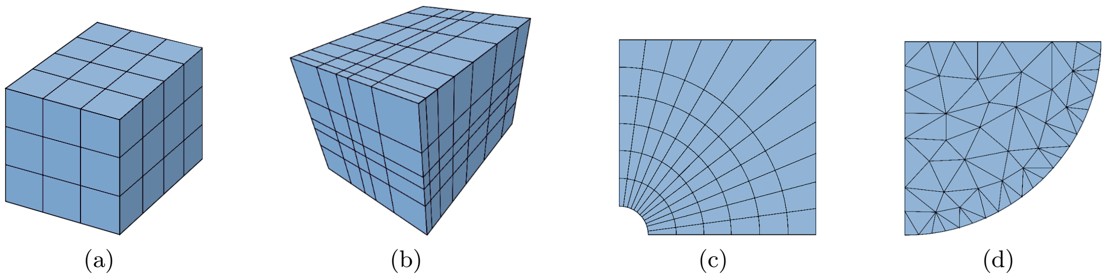


Figure 2.1: Data meshes: (a) Uniform, (b) rectilinear (c) curvilinear and (d) unstructured.

The most commonly used unstructured 3D data representation is the tetrahedral mesh due to its flexibility and ability in filling any arbitrary 3D complex geometries [88]. This format is generally used in CAD, finite element method (FEM) and simulation. Figure 2.2(a) shows an instance of this representation. While the mesh model represents a 3D object by a collection of polyhedra, the voxel model is a discrete collection of uniformly distributed unit cubes in 3D space [65]. This uniform structured model is commonly used to represent sampled medical and scientific data (see Figure 2.2(b)).

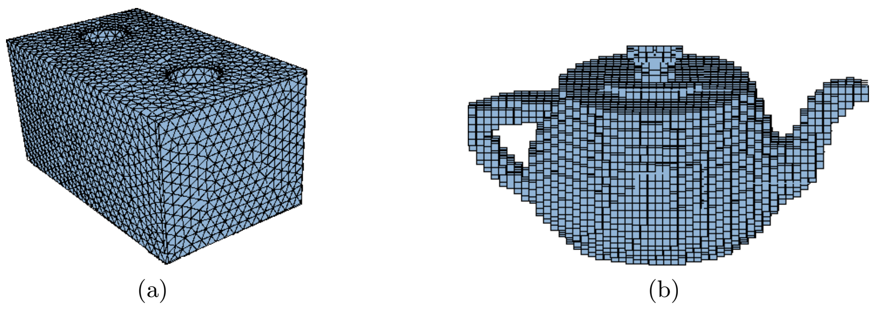


Figure 2.2: (a) A tetrahedral mesh model and (b) a voxel model.

Another way of representing the topological information of 3D objects in order to be processed by a computer is offered by Morse theory and Reeb graphs. Morse theory [54, 96] can be thought of as a generalization of the classical theory of critical points (minima, saddles, and maxima) of smooth functions on Euclidean spaces. Morse theory states that for a generic function defined on a closed compact manifold (e.g. a closed surface) the nature of its critical points determines the topology of the manifold. Reeb graphs [57] are symbolic representation of a certain subset of Morse functions, they represent the configuration of critical points and their relationship, and provide a way to understand the intrinsic topological structure of an object.

In this thesis we consider uniform meshes, because our objective is to contribute particularly in the bioengineering field where the data acquisition devices produce rectilinear data sets by sampling. Moreover, real samples used to compute structural parameters are composed of two phases (solid and porous space), therefore, they are represented as binary volumes.

In most of the reported literature, the operations to study binary volume datasets are performed directly on the classical voxel model. However, in the field of volume analysis and visualization, several alternative models have been devised for specific purposes. Hierarchical structures such as octrees and kd-trees have been used for Boolean operations [134], connected-component labeling (CCL) [34], and thinning [118] [182]. Octrees are used as a means of compacting regions and getting rid of the large amount of empty space in the extraction of isosurfaces [179]. Their hierarchy is suitable for multi-resolution when dealing with very large data sets [7, 77], as well as to simplify isosurfaces [167]. Kd-trees have been used to extract two-manifold isosurfaces [50].

There are models that store surface voxels, thereby gaining storage and computational efficiency. The semi-boundary representation affords direct access to surface voxels and performs fast visualization and manipulation operations [51]. Certain methods of erosion, dilation and CCL use this representation [159]. The slice-based binary shell representation stores only surface voxels and is used to render binary volumes [69].

A binary voxel model represents an object as the union of its foreground voxels and its continuous analog is an OPP [76]. The Extreme Vertices Model (EVM) and the Ordered Union of Disjoint Boxes (OUDB) [1, 3] represent OPP in a compact way. EVM stores only a sorted subset of vertices of the OPP boundary, whereas OUDB keeps a sorted list of boxes that compose the whole object.

OPP have been used in 2D to represent the extracted polygons from numerical control data [108]. Some 3D applications of OPP are: general computer graphics applications such as geometric transformations and Boolean operations [1, 19, 38], virtual MIP without skeleton [126], direct skeleton computation (instead of iterative peeling techniques) [36, 91], and orthogonal hull computation [16, 17]. OPP have been also used in theory of hybrid systems to model the solutions of reachable states [19, 30].

A brief summary of the most common models to represent binary volume datasets reported in the literature is presented below.

2.2.1 Voxel Model

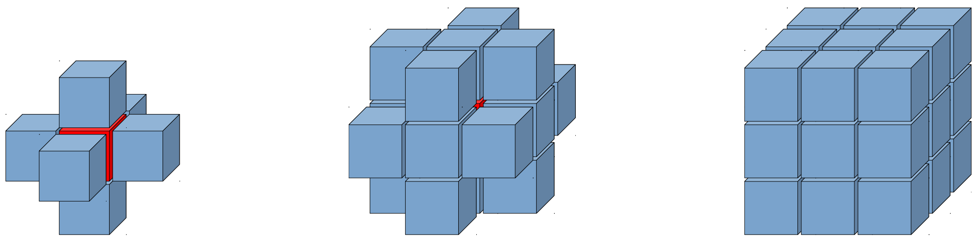
The voxel model [65, 121] is based on a regular decomposition of the 3D space into a set of identical cubic cells called voxels. In a voxel model, voxels are all the same size and their edges are parallel to the main axes. Formally, a voxel model V of size $n_x \times n_y \times n_z$ is defined as:

$$V = \{v_{i,j,k} \mid 0 \leq i \leq n_x, 0 \leq j \leq n_y, 0 \leq k \leq n_z\}$$

where $v_{i,j,k}$ is a voxel in location (i, j, k).

From V, all geometric and topological information can be obtained. On each voxel $v_{i,j,k}$, there is a set of associated values. For a binary voxel model, the associated value of its voxels is limited to $v_{i,j,k} \in \{0, 1\}$, where 0 corresponds to the background and 1 to the foreground.

Given a voxel v, the voxels that surround it form its local neighborhood. Three kinds of adjacency relations are defined between voxels: 6, 18 and 26-adjacency. Two voxels are 6-adjacent if they share a face, 18-adjacent if they share an edge or a face, and 26-adjacent if they share at least a vertex (see Figure 2.3).



(a) 6-adjacency (b) 18-adjacency (c) 26-adjacency

Figure 2.3: Kinds of adjacency of a voxel with its local neighborhood.

An adjacency pair (m, n) defines the adjacency of a binary volume dataset, meaning that the foreground is m-adjacent and the background is n-adjacent. Using some adjacency pairs leads to paradoxes making the choice of foreground and background to become critical, and several times it is not clear what is the foreground and what is the background [73, 80]. Therefore, proper adjacency pairs that avoid paradoxes are useful, in 3D these adjacency pairs are (6, 26) and (26, 6) [76].

A binary volume model is manifold (well-composed) if it lacks the shapes shown in Figure 2.4 (middle and right), modulo reflections and rotations [79]. However, general binary volumes with adjacency pairs (6, 26) or (26, 6) are non-manifold as 26-adjacency allows non-manifold shapes.

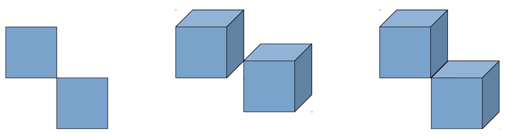


Figure 2.4: Non-manifold 2D (left) and 3D (middle and right) configurations.

Algorithms for the voxel model are straightforward to implement. However, just because of the size of the source models, it has the drawbacks of the loss of geometric information and high memory and computational power requirements [67]. To reduce the memory footprint and the computation time of the algorithms, many alternative models have been proposed, such as hierarchical data structures and boundary representations.

2.2.2 Hierarchical Structures

Bintree. A bintree [136] begins with a cubic cell encompassing the represented subspace, and a recursive subdivision (by an axis-aligned hyperplane that intersects the interior of that subspace) into two equal parts of space is made, recording which parts are empty (outside) and which are solid (inside). Those intermediate cells, which contain part of the data are subdivided. The process is repeated until all cells are black or white, or a limit previously established of subdivisions is reached.

Binary Space Partition (BSP) tree. BSP trees [100, 158] are similar to bintrees except that the position and direction of the subdivision hyperplanes are usually selected following an optimization heuristic (i.e., they are not axis-aligned), e.g., for improving tree balance or for reducing the number of partitions.

Octree. Like bintrees, an octree [137, 179] begins with an initial cube, and a recursive subdivision into eight sub-cubes of equal size is made. This model is one of the most popular and particularly appropriate for representing sample data volumes common to scientific visualization. Similar to other hierarchical models, non trivial geometric transformations such as translation, scaling or rotation of the object may need to recompute the model. However, octrees are quite apt for divide-and-conquer operations or fast hidden surface removal due to their spatially ordered nature.

Kd-tree. A kd-tree [15, 31] is a special case of bintree, where each level is asymmetrically divided in alternate directions according to a discriminant (e.g., X, Y or Z-coordinate). The node's pivotal coordinate is placed where needed for an optimal subdivision, thus creating two child cells with different sizes. If the plane separator position is fixed in the middle position, it is equivalent to a bintree.

In the aforementioned structures, cell subdivision is repeated until a satisfactory level of refinement or until achieving the maximum level of recursion. Valid terminal nodes (also known as leaf or terminal nodes) include white nodes (completely outside the object) and black nodes (completely inside it). Whenever an octant cannot be represented as a valid terminal node, it is denoted as a gray node. Some models, such as extended octrees [21], allow these heterogeneous gray nodes which cut the object boundary and contain complex geometry.

Constructive Solid Geometry (CSG). CSG [121] is a well-known representation model where simple primitive solids (boxes, spheres, general half-spaces, etc.) are combined by using Boolean operations included directly in the representation. An object is stored as an ordered tree with operators at the internal nodes, and simple primitives at the leaves. The drawback of CSG representations is that they are not unique [37]. Since boundary representations are usually more used in CAD applications, some techniques have been devised to improve the efficiency of boundary algorithms in CSG, like the active zones tool [130] to detect and eliminate redundant nodes in CSG trees, trimming expressions of faces in CSG models to identify its contribution to the boundary of the object [129] and techniques for avoiding the production of inconsistent boundary models [14].

Extended Convex Differences Tree (ECDT). ECDT [119], a generalization of the Convex Differences Tree [120], is a representation for n-dimensional polyhedra. It can be viewed as a CSG tree with some restrictions that lead to a simpler tree than the original CSG and more efficient operations. ECDT represents an object by a tree where every node holds a convex bound to the set which it represents (not necessarily the convex hull). The drawback of ECDT is the treatment of the union operation when performing Boolean operations directly.

2.2.3 Boundary Representations

When working with binary volume datasets, it is not necessary to keep the exact scanned density values. Therefore, boundary models are also adequate to represent these kind of datasets. The basic model for representing the polygonal surface of an object is the Boundary Representation (**B-Rep**) model [98] that keeps explicitly all of the relationships between geometric elements such as vertices, edges and faces. However, there are many proposed models for representing the object boundary in a more compact way.

Chain Codes. The Freeman chain code [39] is the most used type of chain code. It represents the boundary of a binary 2D object with a sequence of vectors of unit length and a set of eight possible directions. On the digital grid, encoding is based on the fact that successive contour points are adjacent to each other. Usually 4-connected and 8-connected grid are used, then, the chain code is defined as the digits from 0 to 3 or 0 to 7, assigned to the 4 or 8 neighboring grid points in a counter-clockwise sense [138]. The stated objectives of chain codes are to represent black and white 2D images in a lossless, compact and easy to process manner.

Semi-boundary (SB) and Shell. SB is a data structure which stores in a concise way the boundary and the interior information. This representation scheme stores only the boundary voxels of the volume keeping the information of the interior voxels in an implicit way. It requires seven to eight times less storage space and achieves three to five times faster computation than the voxel model [164], and allows fast visualization and manipulation operations with a set of ad-hoc operations specially designed to exploit the conciseness of the model [127]. Shell representation [165] has the same indexing scheme used in the SB representation, but the set of voxels, which belongs to the list that contains all the boundary voxels of the volume, was redefined in order to represent objects with fuzzy boundaries. A shell consists of those voxels that have an opacity value above a given threshold.

Cell-boundary. Cell-boundary [82] is also a very similar representation to SB, which consists of a set of boundary cells with their voxel configurations, so, the points of the sample in SB become vertices of the cells in the cell-boundary representation.

EVM and OUDB. EVM [1, 3] is a very concise representation scheme in which any OPP can be described using a subset of its vertices: the extreme vertices. EVM is actually a complete solid model, it is an implicit B-Rep model, i.e., all the geometry and topological relations concerning faces, edges and vertices of the represented OPP can be obtained from the EVM [125]. This representation model has been used to prove the suitability of OPP as geometric bounds in CSG [2]. OUDB [1, 124] is a special kind of spatial partitioning representation derived from EVM, where an OPP is decomposed in a list of disjoint boxes. OUDB is axis-aligned like octrees and bintrees, but the partition is done along the object geometry like BSP.

EVM and OUDB models reduce the complexity of some OPP operations. In this thesis, the most used representation models are EVM and a new model derived from OUDB, CUDB. Therefore, EVM and OUDB are reviewed in more detail in the next sections.

2.2.4 Extreme Vertices Model

In the following we review the main definitions and properties of the EVM used in this thesis. For a more extensive explanation see [1] and [3].

EVM is a very concise representation scheme for OPP. Orthogonal polyhedra are two-manifold polyhedra with all their faces oriented in the three main axes. They are also named rectangular or isothetic. OPP are regular orthogonal polyhedra with a possible non-manifold boundary [79]. OPP are a restricted class of polyhedra considering its geometry, but concerning topology OPP are general, as they present any number of shells, cavities and holes. General OPP are OPP with vertex coordinates having any value in $\mathbb{R}^3$. Polycubes are a subset of OPP all of whose vertices have integer coordinates, formed by joining one or more equal cubes (voxels) face to face.

EVM stores the extreme vertices (EV) with no additional information, such as topological relations among vertices, edges, or faces, since all these elements can be computed from them. Therefore, the storage requirement for an OPP P in its EVM representation is $O(n)$, n being the number of EV, and $n \leq v$, where v is the total number of vertices.

Consider an OPP P, embedded in a 3D grid in such a way that all the vertices of P coincide with the grid vertices. A vertex v, is classified according to the color black or white of its eight surrounding voxels. There are $2^8 = 256$ combinations, which applying reflections and rotations may be grouped into 22 configurations. Eight of these configurations correspond to an EV and are those with an odd number of black and white surrounding voxels (see Figure 2.5) while configurations with an even number of black and white voxels do not correspond to EV [1].

Definitions

Let P be an OPP:

- *Axis-prefixable:* A linear element is axis-prefixable when it is parallel to a prefixed axis. A planar element is axis-prefixable when it is perpendicular to the prefixed axis.

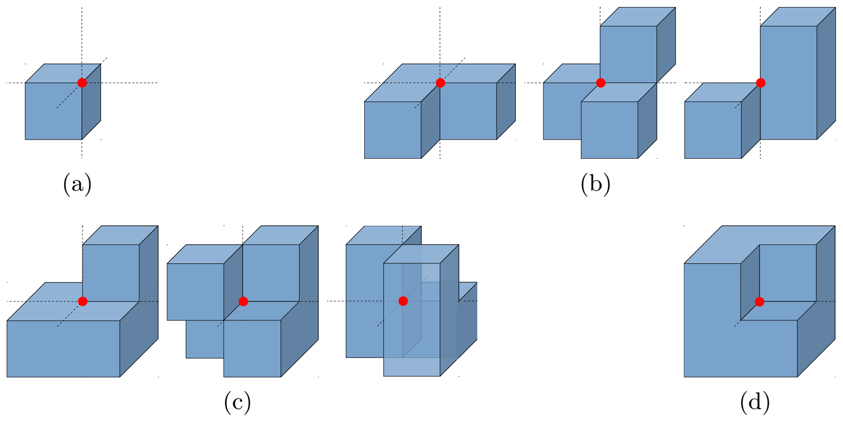


Figure 2.5: The eight 3D configurations where a surrounded vertex is extreme vertex. (a) one voxel, (b) three voxels, (c) five voxels, and (d) seven voxels surrounding a vertex.

- *ABC-sorting:* Let Q be a finite set of points in $\mathbb{R}^3$. The ABC-sorted set of Q is the set resulting from sorting Q according to coordinate A, then to coordinate B, and then to coordinate C. Thus, six possible ABC-sorted sets can be defined in 3D: XYZ, XZY, YXZ, YZX, ZXY, and ZYX.

- *Brink:* is the maximal uninterrupted segment built out of a sequence of collinear and contiguous two-manifold edges of P. Brinks are axis-prefixable.

- *Extreme vertices:* are the ending vertices of all brinks in P. $EV(P) \subseteq V(P)$. $V(P)$ being the set of all vertices of P.

- *Extreme Vertices Model:* is the ABC-sorted set of all EV of P. Any OPP is described by its (and only its) set of EV.

- *Cut (C):* is the 2D OPP whose EVM-representation is the set of EV of P lying on a plane perpendicular to a main axis of P. In 2D, a cut is the set of brinks lying on a line parallel to a main axis. Cuts are axis-prefixable and $C_i(P)$ refers to the i-th cut of P.

- *Slice:* $slice_i(P)$ is the prismatic region between two *consecutive cuts* $C_i(P)$ and $C_{i+1}(P)$. P can be expressed as the union of all its slices in a certain orthogonal direction. Hence, considering the orthogonal direction A, $P = \bigcup_{i=1}^{n-1} slice_i(P)$, where n is the number of different A-coordinates in P (i.e., the number of cuts). A slice from a 3D OPP is a set of one or more disjoint prisms, while a slice from a 2D OPP is a set of one or more disjoint rectangles.

- *Section (S):* is the resulting polygon (or set of polygons) from the intersection between a slice of P and an orthogonal plane parallel to the cuts. The base of each slice is a section. $S_i(P)$ refers to the i-th section of P between $C_i(P)$ and $C_{i+1}(P)$. In 2D, a section is the resulting line (or set of lines) between a slice of P and an orthogonal line parallel to the cuts.

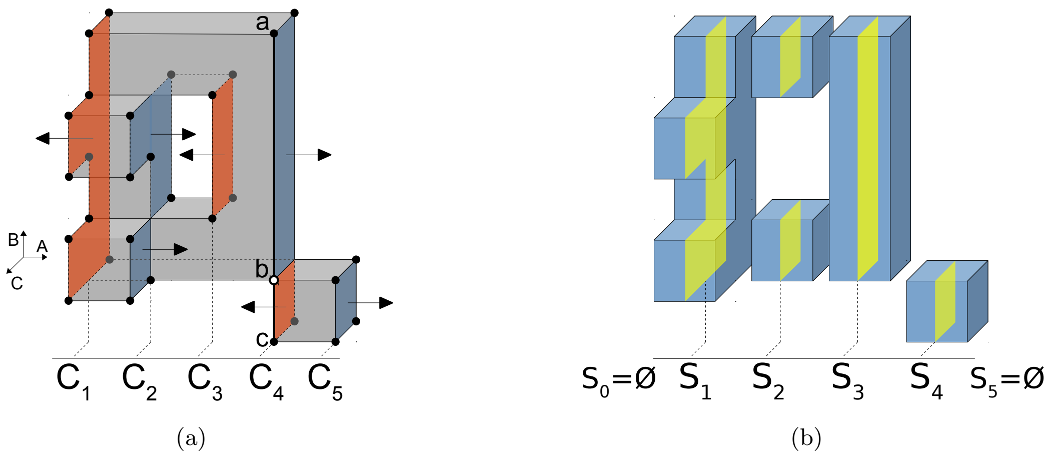


Figure 2.6: An EVM-represented (ABC-sorted) OPP. (a) EV marked with black dots, cuts decomposed in FD (blue) and BD (orange), normal vectors represented with arrows; a vertical brink from vertex **a** to **c** is marked showing that these vertices are both EV while vertex **b** is non-EV. (b) Object's slices and sections highlighted in blue and yellow respectively.

Figure 2.6(a) shows an OPP with all its EV, cuts, slices and sections.

Properties

All of the following properties are proved in [1]. Let P be an OPP:

Property 1: Coordinate values of non-extreme vertices can be obtained from EV coordinates.

Property 2: Sections can be computed from cuts and vice versa:

$$\overline{S_0(P)} = \overline{S_n(P)} = \emptyset, \quad \overline{S_i(P)} = \overline{S_{i-1}(P)} \otimes^* \overline{C_i(P)}, \quad \forall i = 1, \ldots, n-1 \tag{2.1}$$

$$\overline{C_i(P)} = \overline{S_{i-1}(P)} \otimes^* \overline{S_i(P)}, \quad \forall i = 1, \ldots, n \tag{2.2}$$

where n is the number of cuts and $\otimes$ denotes the XOR operation. Overline symbolizes the project operator, that projects a d-dimensional set of vertices lying on an orthogonal plane, like a cut or a section, onto the corresponding main plain, discarding their d-th coordinate. The star exponent * denotes a regularized Boolean operation. Regularized Boolean operations are needed to ensure 3D homogeneity [160].

Property 3: The computation of any cut (Equation 2.2) can be rewritten by expressing the $\otimes^*$ operation as the union of differences and any cut can be decomposed into its *forward difference (FD)* and *backward difference (BD)*. For $i = 1...n$:

$$\overline{C_i(P)} = (\overline{S_{i-1}(P)} -^* \overline{S_i(P)}) \cup^* (\overline{S_i(P)} -^* \overline{S_{i-1}(P)}) \tag{2.3}$$

$$\overline{FD_i(P)} = \overline{S_{i-1}(P)} -^* \overline{S_i(P)} \tag{2.4}$$

$$\overline{BD_i(P)} = \overline{S_i(P)} -^* \overline{S_{i-1}(P)} \tag{2.5}$$

The application of this property allows us to obtain the oriented faces of the OPP as well as all the hidden non-extreme vertices. $FD_i(P)$ is the set of $C_i(P)$ faces whose normal vector points to the positive side of the coordinate axis perpendicular to $C_i(P)$, and $BD_i(P)$ is the set of faces whose normal vector points to the negative side (see Figure 2.6).

Property 4: Let P and Q be two d-dimensional ($d \leq 3$) OPP, having $EVM(P)$ and $EVM(Q)$ as their respective models, then:

$$EVM(P \otimes^* Q) = EVM(P) \otimes^* EVM(Q) \tag{2.6}$$

Property 5: Let P and Q be two OPP such that $P \cap^* Q = \emptyset$, having $EVM(P)$ and $EVM(Q)$ as their respective models, then:

$$EVM(P \cup^* Q) = EVM(P) \otimes^* EVM(Q) \tag{2.7}$$

Property 6. Let P and Q be two OPP such that $P \supseteq Q$, with $EVM(P)$ and $EVM(Q)$ as their models, then:

$$EVM(P -^* Q) = EVM(P) \otimes^* EVM(Q) \tag{2.8}$$

Property 3 provides proof that EVM is a complete B-Rep model and, therefore, it represents OPP unambiguously [172]. Property 4 means that the XOR operation works in 0D, because it applies directly to the EV of the model. Therefore, sections are obtained from planes of vertices and vice versa by applying the XOR operation to the extreme vertices (Property 2).

General EVM Boolean operations such as union, intersection and difference are carried out by applying recursively (in the dimension) the same Boolean operation over the 2D OPP sections. The base case performs this operation in 1D. XOR operation is even faster, i.e., Property 4 is a simple point-wise XOR without section computation.

Although EVM has been extended to nD [112], in this thesis we deal with dimension $n \leq 3$.

Methods

EVM has been implemented as an object with a set of methods. This section enumerates the main methods often referenced and required in the algorithms presented in this thesis. Let P, C and br be EVM objects (abstract data types), and dimType and sortingType two predefined enumerated data types:

- *P.createEVM*(dimType *dim*, sortingType *ABC*): Creates an empty EVM object P of dimension *dim* and sorting *ABC*.
- *P.getNextCut*(): Returns the current cut of P and its A-coordinate, and set the pointer at the following cut.
- *P.getBrink*(): Returns the current brink of a 1D P and set the pointer at the next brink.

- $P.getDimension()$: Returns the dimension of P.
- $P.getNEV()$: Returns the number of extreme vertices of P.
- $P.getSorting()$: Returns the ABC-sorting of P.
- $P.getArea()$: Returns the area of the 2D OPP P.
- $P.getVolume()$: Returns the volume of the 3D OPP P.
- $P.insertCut($EVM C, real $coord)$: Inserts the cut C into P at A-coordinate $coord$.
- $P.readBrink()$: Returns the extreme points ($P0$ and $P1$) of a brink.
- $P.setSorting($sortingType $ABC)$: Sorts the EV of P according to the sorting ABC.

2.2.5 Ordered Union of Disjoint Boxes Model

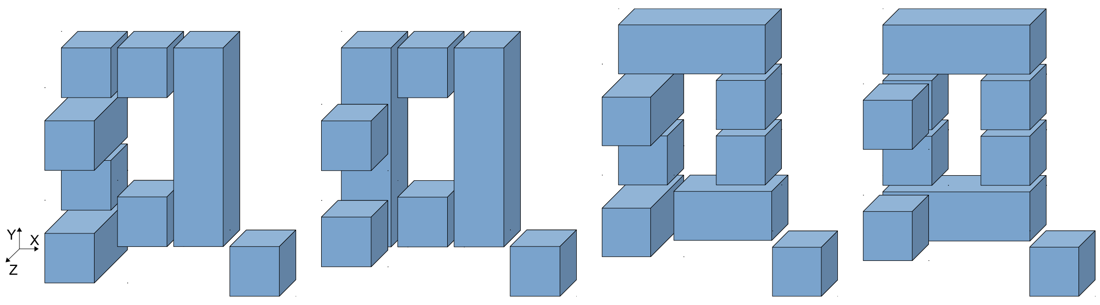
An OPP P can be represented as a list of disjoint boxes. From an ABC-sorted EVM, the ABC-ordered OUDB for P can be obtained. This model is the set of boxes obtained by [1]:

1. Splitting P at every internal cut of P perpendicular to the A-axis $C_i(P)$, $i = 2, \ldots, n-1$, where n is the number of cuts. Thus, obtaining an ordered sequence of A-slices, i.e., $P = \bigcup_{i=1}^{n-1} slice_i(P)$. See Figure 2.6(b).

2. Splitting each A-slice at every one of its internal cut perpendicular to the B-axis, thus, obtaining a sorted sequence of disjoint boxes, i.e., $slice_i(P) = \bigcup_{j=1}^{n_i-1} Box_{i,j}(P)$.

Then, an OPP P is expressed in the OUDB model as:

$$P = \bigcup_{i=1}^{n-1} \bigcup_{j=1}^{n_i-1} Box_{i,j}(P) \tag{2.9}$$



(a) 8 boxes (b) 7 boxes (c) 8 boxes (d) 9 boxes

Figure 2.7: Four possible OUDB decompositions for the OPP in Figure 2.6(a). (a) XYZ-OUDB. (b) XZY-OUDB. (c) YXZ-OUDB. (d) YZX-OUDB.

There are six different ABC-OUDB models for a given OPP, which correspond to each of the ABC-sorted EVM. Their corresponding sets of disjoint boxes are generally different. Figure 2.7 shows four possible OUDB decompositions for the OPP in Figure 2.6(a).

As OUDB contains a lesser number of elements than the voxel model (boxes instead of voxels), it is very efficient in some tasks that require traversing the model, such as volume and center of gravity computation [12], and CCL [124].

2.3 Structural Parameters

Geometrical and topological representations of the internal structures of samples used in Bio-CAD (bones, biomaterials, rocks and other material samples) are necessary to evaluate their physical properties. Such samples have two disjoint spaces, the pore space and solid space, thus, they can be represented with binary volume models. The pore space, in turn, is made up of a collection of pores that can be intuitively defined as local openings interconnected by narrow apertures called throats (or necks) that limit the access to a larger pore [43] (see Figure 2.8).

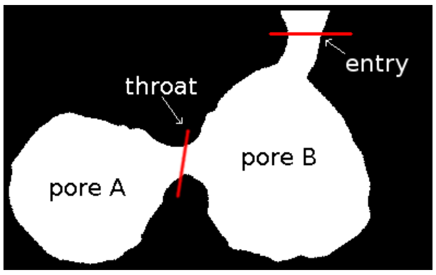


Figure 2.8: Two pores separated by a throat.

In biological experiments, some objects cannot be directly used as experimental samples. Thus, life sciences in general use different alternatives of experimentation. These experimental systems can be distinguished into *in-vivo* representing the function of entire organisms, and *in-vitro* models representing the function of certain parts outside of an organism. *In-silico* experimentation refers to the use of mathematical models in experiments performed on computers [114]. In this case, volume datasets are obtained with non-invasive capturing methods such as CT, nCT, μCT, MRI, NMR, SPECT, PET, etc. Then, a voxel model is reconstructed and segmented in order to differentiate the two regions in the sample. This model is then used to analyze and visualize the desired structural parameters.

Usually, porous datasets are considered adequate for analysis if they have their holes homogeneously distributed along the sample, and the pore size is notably smaller than the dataset size but clearly larger than the voxel's length [43]. That is, they contain complete holes and the voxel resolution is high enough to clearly define their shape. For instance, Figure 2.9(a) shows a real cylindrical sample of hydroxylapatite. Figure 2.9(b) and 2.9(c) are a slice of the pore space before and after the binary segmentation process respectively.

The study of porous materials properties is of great utility in several disciplines. In medicine, they are used to evaluate the degree of osteoporosis and the adequacy of synthetic biomaterial

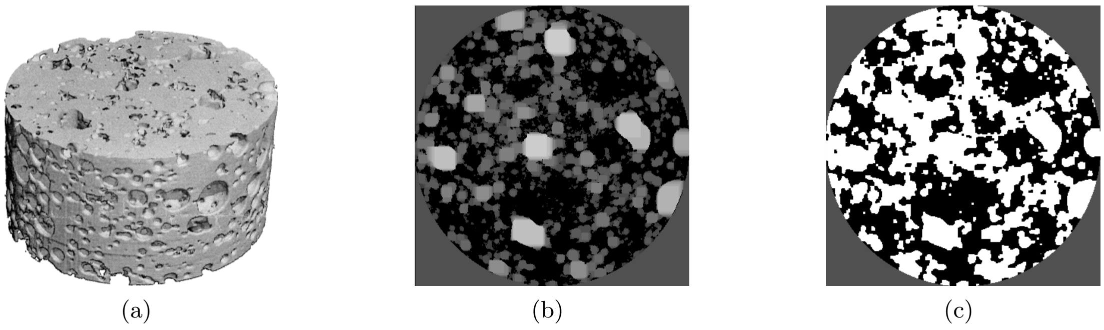


Figure 2.9: Example of biomaterial segmentation: (a) Real sample of hydroxylapatite., (b) density values and (c) binary segmentation (in white the pore space).

implants for bone regeneration, among other applications. Bone regeneration occurs in the cavities of the implants, where blood can flow. Biomaterial implants can be designed as tissue scaffolds, that is, extracellular matrices onto which cells can attach and then grow and form new tissues [152, 153]. In geology, the porous structure of a rock is related to its oil-bearing and hydrological properties [147]. Likewise, brine inclusions in sea ice can be formalized in terms of their morphology and connectivity [46]. Silica sands need to have a high sphericity, as where the more round and spherical is the particle, the more resistant that particle is to crushing or fragmenting [143]. In engineering, the durability of cementitious materials is associated with certain mechanical and transport properties that can be evaluated based on the properties of the pore space [150].

There are many structural parameters for describing porous materials. Some of them have been traditionally computed using 2D methodologies and their results extended to 3D by means of stereologic techniques, while other parameters can be computed directly from the volume dataset. We focus on parameters and methodologies that use volume datasets. Some of this parameters are simply the volume of different phases [163] or the contact surface between them, but others are more complex. A non-exhaustive list of the more common structural parameters that have been found in the literature is briefly presented below. It is mainly based on the set of parameters computed from bones and biomaterial samples in [25] and [168].

- **Porosity** is a parameter linked to both the mechanical behavior of a material and the volume available. *Total porosity* is defined as: $poreVolume/totalVolume$, where $totalVolume = solidVolume + poreVolume$.
- **Pore interconnectivity** is the parameter that marks the degree to which pores are connected to each other and to the exterior of the sample. It can be computed as: $nonIsolatedPoreVolume/poreVolume$.
- **Pore-size distribution** is a useful tool that helps to interpret the characteristics of samples by allowing users to observe most common pore diameter ranges as peaks in a plotted graph. It is commonly represented with just a histogram; however, a complete pore graph is normally devised, as it provides more information and is the basis for performing

permeability studies. For instance, it is used to evaluate the suitability of biomaterials to form new tissues, as well as to understand fluid distribution in gas- or oil-bearing rocks.

- **Connectivity** [103] is a topological property related to the genus of the solid space that measures the level of interconnectivity among elements, and is an indicator of the biomechanical characteristics of bone or other materials. It can be computed as a global measure of the binary volume or extracted from the pore graph model.
- **Fluid flow permeability** [5] is a measure of the ability of a material to transmit fluids. A high permeability points to an easy circulation for fluids at low pressures. Usually, this measure requires previously devising a graph representation of the pore space.
- **Orientation** of a shape may be defined by rotation angles around a set of orthogonal axes [56]. Preferred orientation arises when the shape is oriented preferentially in a certain directions or set of directions [133]. Orientation is related to anisotropy [103], which refers to the exhibition of different values of a property when measured in different directions.
- **Sphericity and Roundness.** Sphericity [176] is a measure of how spherical is a particle and is independent of its size. The more compact a particle is, the more closely it resembles a sphere. Roundness [75] is the measure of the sharpness of a particle's edges and corners. Sphericity and roundness are ratios and, therefore, dimensionless numbers. They can be applied in geology, where it is important to classify particles according to their shape.

Structural parameters such as porosity and pore interconnectivity can be straightforward computed once the solid and pore space volumes has been defined. More interesting parameters that require more complex methods to be analyzed are the connectivity and those related to the morphology of the pore space that allow us to compute the pore-size distribution. Previous work to compute these structural parameters is reviewed below.

2.3.1 Porosimetry

Pore-size distribution is usually computed with mercury intrusion porosimetry (MIP), an experimental method based on the capillary law governing liquid penetration into small pores. In this technique, mercury is intruded into the sample at increasing pressures, causing the fluid to flow through smaller apertures. A lab porosimeter gives pairs of applied pressure and intruded volume, and the Washburn's equation can be used to obtain a pore-size histogram. This equation assumes that pores have a circular cross-section and relates pressure to pore diameter.

$$D = -\frac{4W\gamma cos\theta}{\rho} \tag{2.10}$$

where D is the diameter, ρ the applied pressure, γ the surface tension, θ the contact angle and W the Washburn constant.

MIP is an in-vitro experiment performed in a wet lab and is subject to some of the common problems associated with this kind of experiment. It entails a costly analysis, uses toxic products, delicate equipment, and requires a trained lab technician. Moreover, with MIP, samples are deformed during the experiment, due to the pressure applied, and cannot be reused. As they are filled with mercury, they thus become toxic waste.

In-silico experimentation does not have the drawbacks discussed above, but is not free of problems, which in this case, are related to the device's resolution and the segmentation process. Due to the different natures of the experiments and the possible causes of error, it is difficult to compare experimental and *in-silico* results [171].

Virtual porosimetry for general porous materials simulates MIP at incremental pressures [43]. All separated components of the pore space filled at a given pressure are considered pores and labeled with the diameter corresponding to the applied pressure. This is a flood-fill methodology that uses a previously computed 2D (surface) skeleton as a guide to simulate mercury intrusion from the entry points into the pore space. Moreover, the skeleton is labeled at each point with the corresponding distance (the maximum radius) used to allow or stop the mercury intrusion simulation, i.e., when this radius is smaller than the radius for the current pressure, the simulated intrusion at this pressure stops. Regions corresponding to these smaller radii are the throats. This method is applied to biomaterial samples [170] and soil samples [32]. The latter uses the obtained pore network to compute the permeability. Several methods define the initial set of entry points as those pore voxels which are connected to exterior; however, it can be predefined to allow more freedom to the user in the simulation.

There are other approaches to compute the pore-size histogram and pore graph too. Some are related to the shape-analysis discipline and also use a 2D skeleton as a tool to devise the shape and size of pores. These approaches are heuristic methods that cover the pore space with overlapping spheres, so that the pores are computed as the unions and differences of maximal spheres centered at skeleton points. These methods can be applied to sand samples [147], bone scaffolds [25], and biomaterial samples [169].

Other approaches based on the 1D (curve)-skeleton computation detect throats as the absolute minima of the skeleton. Thus, pores are defined as the regions limited by throats and solid space [85]. A graph is obtained directly from the 1D skeleton, in which nodes correspond to pores and edges to throats [84]. Alternatively, the minimal cost paths connecting boundary points can be computed instead of skeletons to allow methods based on porosimetry, as well as sphere positioning, to be applied [141].

Another technique for computing pore-size histograms is granulometry. This methodology consists in the application of successive morphological openings and does not require the computation of a skeleton [26, 59, 144, 174]. A specific set of discrete spheres with diameters D such that $S(D) \subset S(D+1)$ can be used, thereby ensuring that a larger ball will never reach a cavity in which a smaller ball cannot enter [59]. Physical MIP has been compared with a granulometry-based method [26].

Throats can also be detected based on the negative Gaussian curvature of the surface. This method is used to decompose the solid space into single particles and to compute mechanical properties such as grain size and coordination number (number of contacts per grain) [157].

2.3.2 Fluid Flow Permeability

The flow of incompressible fluids in porous media can be modeled using commonly the Navier-Stokes equations [106] and the Lattice-Boltzmann method [18]. A direct simulation of the Navier-Stokes equations in a percolation porous structure can be used to study fluid-flow behavior [5]. MIP simulation can also be addressed as a fluid-flow method with a Lattice-Boltzmann method [62]. However, most approaches do not rely on the actual dynamic process of mercury intrusion, but rather simulate it in a static way, directly measuring the pore space geometry to obtain the pore graph. Moreover, permeability and other hydraulic properties can be computed by applying Kirchoff-based methods to the obtained pore graph [32]. Geometric and topological descriptors are also used to enhance the estimation of material permeability [166], characterizing the pore space with a Reeb graph [57] instead of a skeleton.

2.3.3 Connectivity

Connectivity and genus are related terms used in many scientific fields, for instance, in Bio-CAD, the connectivity is related to biomechanical properties and is used to measure the strength of bones (osteoporosis) or the quality of the biomaterials designed to repair them. Some methods obtain the connectivity as a global property, treating all the porous space as a single object and analyzing the binary voxel model of the sample.

The connectivity of a volume model can be estimated from the Euler-Poincaré characteristic, χ, which can be computed from a voxel model with the following expression [104]:

$$\chi = n_0 - n_1 + n_2 - n_3 \tag{2.11}$$

where n_0, n_1, n_2 and n_3 are, respectively, number of vertices (points), edges (linear elements), faces (surfels) and voxels of the voxel model. This expression can be applied to several adjacency pairs [161]. From the theory of homology, the Euler-Poincaré formula relates χ with the Betti numbers h_i [92]:

$$\chi = h_0 - h_1 + h_2 \tag{2.12}$$

where h_0, h_1 and h_2 are, respectively: number of connected components, connectivity and number of isolated cavities. h_0 and h_2 are usually computed using CCL-based methods over voxels, polyhedron faces or triangles, depending on the model used. Then, the connectivity h_1, which is related to the genus, can be computed from χ, h_0 and h_2 using Expression 2.12.

In the solid modeling field, χ can also be computed from a polyhedron using the following expression [89]:

$$\chi = V - E + F - R = 2(S - g) \tag{2.13}$$

where V, E, F and R are, respectively: number of vertices, edges, faces and internal rings of faces. S is the number of shells (number of sets of connected faces) and g is the genus. For the special case of triangulated surfaces, χ can be expressed as:

$$\chi = V - E + T \tag{2.14}$$

V, E and T being the number of vertices, edges and triangles of the mesh. A triangle mesh can be extracted from a voxel model to approximate a given isosurface separating interior and exterior voxels. This mesh can be obtained with the Marching Cubes algorithm [87] or one of its derived methods that have sought to improve the topological correctness and accuracy of the surface representation [86].

Based on the fact that a binary volume dataset can be represented in a compact way by an orthogonal pseudo-polyhedron [68], a previous approach [13] computes χ and the genus of a binary volume dataset using expression 2.13. In this method, the binary volume is represented with EVM. As general datasets can present non-manifold configurations, in order to correctly apply the aforementioned expression, the EVM-representation needs to be converted into an homotopic manifold analog first. However, this approach has proved to be more efficient than methods based on voxel models and triangle meshes.

Expression 2.13 and 2.12 are used to compute the connectivity of a triangular mesh representing some adenine properties in the biochemistry field [74]. In isosurface extraction, the topology-preservation is sometimes a desirable property that can be evaluated by computing χ [140]. The method based on Expressions 2.11 and 2.12 is used to evaluate the osteoporosis degree of mice femur [90], human vertebrae [104] or to evaluate hydraulic properties of sintered glass [174].

Skeletons are also used as a shape-analysis tool in the computation of connectivity. A 2D skeleton is used to devise the so-called plate/rod model [113, 149], in which the model is segmented into linear and surface elements, and connectivity is measured as the relative number of plate and rod elements. Other approaches [115, 116] use a 1D skeleton and apply a line skeleton graph analysis (LSGA) based on the strong relationship between the number of loops in the graph and connectivity.

2.3.4 Orientation

The basic method to describe the orientation of a 3D object is by means of the eigenvectors of the covariance matrix associated to the point set of the object. This point set forms a cloud and has some statistical distribution characterized by the mean and the covariance matrix. The mean describes the center of mass and the covariance matrix contains information about how the cloud is approximately spread out. Eigenvectors of that matrix give the orientation along which the cloud has maximum and minimum statistical spread [47].

An oriented bounding box (OBB) is a box which may be arbitrarily oriented (rotated), its faces have normals which are pairwise orthogonal [48]. An OBB can be represented with a center point $\mathbf{c}$, three extents $h_1 > h_2 > h_3$, and an orientation specified with three mutually orthogonal unit vectors $\mathbf{v}^1$, $\mathbf{v}^2$ and $\mathbf{v}^3$. An OBB for a set of points can be created from the eigenvectors.

Computation of eigenvectors and eigenvalues to determine the orientation of an object has several applications. For instance, preferred orientations are used to predict stiffness and strength in production of fabric tensors [71] or to observe the mass transport phenomena in ionic crystalline materials [132]. In geology, orientation of many clasts in a soil sample can be collected and compared graphically to provide information about their transport history and the characterization of depositional environments [49].

2.3.5 Sphericity and Roundness

Sphericity and roundness are measures of two different morphological properties. Sphericity is most dependent on elongation, whereas roundness is largely dependent on the sharpness of angular protrusions (convexities) and indentations (concavities) from the object.

Sphericity may be calculated for any 3D object if its surface area and volume are known. Wadell [176] defined the sphericity, Ψ, of an object as the ratio of the nominal surface area (surface area of a sphere having the same volume as the object) to the actual surface area of the object. This ratio is known as true sphericity index.

$$\Psi = \frac{\mathcal{S}_n}{\mathcal{S}} = \frac{(36\pi\mathcal{V}^2)^{\frac{1}{3}}}{\mathcal{S}} \tag{2.15}$$

where $\mathcal{V}$ and $\mathcal{S}$ are the volume and surface area of the object respectively, and $\mathcal{S}_n$ is the nominal surface area. The sphericity index of a sphere is 1 and, by the isoperimetric inequality, any object which is not a sphere will have a sphericity value less than 1.

Since manual measures of $\mathcal{S}$ are very difficult, other indices have been defined. Several methods are based on length measurement of the three representative axes of an object [75]: a (major axis length), b (medium axis length) and c (minor axis length). The next equations are used frequently in geology:

$$\Psi = d_n/a \tag{2.16}$$

$$\Psi = (bc/a^2)^{\frac{1}{3}} \tag{2.17}$$

$$\Psi = c/(ab)^{\frac{1}{2}} \tag{2.18}$$

$$\Psi = (c^2/ab)^{\frac{1}{3}} \tag{2.19}$$

Equation 2.16 is a simplified sphericity index proposed by Wadell [177], where d_n is the nominal diameter (diameter of the sphere having the same volume as the object). The index given by Equation 2.17 is called elliptical volume sphericity [75]. The sphericity index given by Equation 2.18 provides more precision for the computation of other behavioral indices [29]. A more widely accepted sphericity index is given by Equation 2.19 as it correlates highly with the particle settling velocity [148].

Concerning roundness, due to the impracticality of measuring a true 3D roundness index, several methods work with the maximum 2D projection plane (silhouette) of the object looking for a trade-off between accuracy and time.

Roundness ($\mathcal{R}$) was defined by Wadell [176] as the ratio of the average radius of curvature of the corners and edges of the object's silhouette to the radius of the maximum circle that can be inscribed.

$$\mathcal{R} = \frac{\frac{1}{n}\sum_{i=1}^{n} r_i}{r_{max}} \tag{2.20}$$

where r_i is the radius of the i-th corner curvature, n the number of corners, and r_{max} the radius of the maximum inscribed circle. The value of $\mathcal{R}$ is 1 for a perfectly round object and less than 1 for any other object.

Results of this method are reliable but time consuming and very impractical as no definition of curvature was established [131]. In order to improve the time required to estimate the roundness, Krumbein [75] created a chart (see Figure 2.10) showing examples of pebbles for which the roundness of their silhouette has been calculated using Equation 2.20 and grouped them into nine classes.

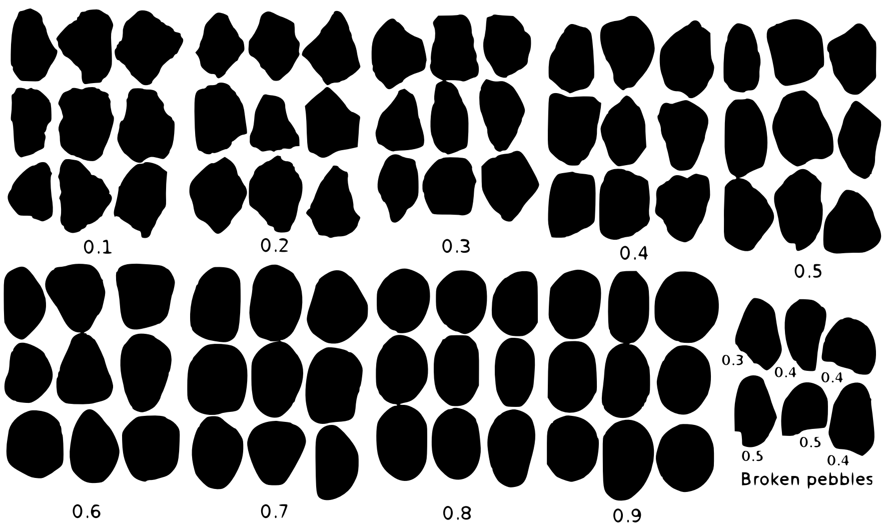


Figure 2.10: Krumbein's chart for visual determination of roundness [75].

After Krumbein, other methods provide estimated values that are linearly correlated with the values given by the Krumbein's chart (KC).

A method based on the Fourier transform [33] makes use of the sum of the amplitudes of the first 24 coefficients of the Fourier transform. To compensate for different size rocks fragments, the coefficients are divided by the zero-th coefficient and the sphericity aspect is eliminated by subtracting the spectrum of the best approximating ellipse from that spectrum. This method shows a correlation of 0.94 with the values of KC.

An alternative approach uses granulometric methods [35]. The ratio between particle's area before and after applying a morphological opening on its silhouette is a roundness index. This method gets a correlation of 0.96 with the values of KC using a circular structuring element of radius equal to 42% of the radius of the largest inscribed circle.

Discrete geometry has been used to calculate the Wadell's original index [131]. The curvature radius at each pixel of the silhouette is calculated with an algorithm that relies on the decomposition of a discrete curve into maximal blurred segments [101], and the radius of the largest inscribed disk is calculated using the distance transform of the silhouette. This method shows a correlation of 0.92 with the original index values of KC.

A true 3D roundness index of gravel shapes is possible to compute using a laser scanner [58]. Based on the idea that the ratio between the volume and surface area of an object reflects the roundness, this method proposes an alternative roundness index. As the ratio $\mathcal{V}/\mathcal{S}$ tends to increase with an increase in size, this ratio is divided by a representative gravel length. Using an ellipsoid as analog of gravel shape, the geometric mean of the three representative axes of the object (a, b, and c) is used as the representative length. The resulting index of the next equation shows a correlation of 0.814 with the values of KC.

$$\mathcal{R} = \frac{\mathcal{V}}{\mathcal{S}(abc)^{\frac{1}{3}}} \tag{2.21}$$

However, the use of a laser scanner to compute the roundness is a very time consuming process for multiple shapes, and not suitable for micro or nano samples.

2.4 Model Simplification

Model simplification has been extensively applied to triangular meshes [24]. Some of these techniques have been extended to tetrahedral meshes and use a methodology to evaluate the approximation error and the quality of the obtained mesh [23]. Methods for level-of-detail (LOD) sequences of triangular and tetrahedral meshes can also be found extensively in the literature [123, 162]. There also exist methods to simplify quadrilateral meshes [50, 155].

In contrast to these methods, that rely on geometric operations such as edge-collapse or clustering, the simplification can follow other strategies. Morphological operators like filleting and rounding, equivalent to opening and closing, can be used to simplify 2D binary images as well as 3D triangular meshes [180]. Alternative representations can be used such as octrees [7, 135, 167] in such a way that the geometry as well as the topology can be simplified, or BSP [61] obtaining a LOD sequence with a decreasing number of nodes. A carving strategy is applied to an octree model [167] as well as to a tetrahedral mesh [55] to simplify the topology. Simplification strategies have also been developed for B-Rep models [151] by removing connected sets of faces, and for point clouds [110].

There has been also an intensive research in surface simplification in a progressive and lossless way, applied specially to triangle meshes [4, 42, 83, 107, 109, 111]. Early methods are called connectivity-driven, as they change progressively the connectivity [4, 107]. Recent approaches for triangle meshes, called geometry-driven, perform an space partitioning by using intermediate structures like kd-trees [42] or octrees [83, 109, 111] in order to code the mesh. In surface simplification the quality of the approximations is evaluated by some distance defined on the points on the surface, regardless of the potentially enclosed volume [6]. The reported compression rates for the geometry-driven techniques are better in general.

2.4.1 Bounding Volumes

Many of the aforementioned approaches provide approximations designed either to preserve the original topology between the original and the simplified object or just to obtain an approximated shape. Nevertheless, in some cases, it is desirable to compute an approximation that is also a bounding volume. Bounding volumes are used in many applications like collision detection [64], ray tracing [45, 178], graphics interaction [63] and volume of interest computation [40].

Several works use classical bounding volumes, such as spheres [70, 41] and the minimum axis-aligned bounding box (AABB) [154, 184]. Nevertheless, other kinds of bounding volumes are used such as convex polytopes with faces oriented in a reduced set of orientations [72] or oriented polytopes that take into account the object's velocity, or dynamic collision detection [28]. Moreover, hybrid bounding volumes are also used. Bounding boxes are enhanced with planes computed from the intersection between two objects [175]. Slab cut balls built from a ball cut by a slab (intersection region between two parallel planes) present a good balance between tightness and testing costs [78]. An oriented bounding box tree, enhanced with bounding spheres at each node, is used to speed up collision detection applying the more efficient sphere test first to eliminate distant objects [22].

Orthogonal polyhedra have been proposed as geometric bounds for constructive solid geometry because general set membership operations with them are very efficient [2]. However, in this work the authors present a Boolean operations algorithm using EVM, but do not present methods to compute bounding orthogonal polyhedra.

There are also simplification methods that generate orthogonal approximations. Iterative orthogonal subdivisions of the AABB of the original model can be made in order to get an predefined number of cuboids (boxes) [10]. Orthogonal polyhedra are used as bounding structures in a coarsening strategy [37], where orthogonal polyhedra are represented with the vertex-list representation [38], which stores a subset of vertices with a weight needed for orientation purposes. The authors present two simplification strategies. One of them, *moving faces*, performs face displacements but fails for objects with holes or more than one connected component. The other strategy, *rectangle pairs* does not have this restriction and performs a partition of the object in boxes, takes them in pairs, obtaining the AABB of this pairs, and finally performs the union of these AABB.

Some methods produce bounding volume hierarchies of polygonal meshes for fast collision detection between massive models [183] and for haptic rendering [105]. Other methods produce a hierarchy of polygonal convex volumes (a shrink-wrap around the object) [61, 94]. Meanwhile, orthogonally convex polygons can be computed as orthogonal hulls for 2D images [17]. An orthogonal polygon is orthogonally convex if any axis-parallel line intersects it in at most one line segment. This problem has been extended to orthogonally convex polyhedra [16]. Such convex polygons, like the convex hull, are not suitable to represent simplified objects with many concavities and holes, i.e., objects that are not close to a convex shape.

Methods that generate bounding approximations evaluate the quality of the approximations as the tightness of the bounding volume with respect to the volume of the original object.

Progressive Simplification with Bounding Volumes

In this section, three methods that produce a progressive LOD sequence of bounding volumes are described in some detail. These methods are compared with the proposed simplification method presented in Chapter 5.

The first method is a technique presented by Samet and Kochut [135] to progressively approximate and compress in a lossless way binary 3D objects represented by pointer-less octrees. This method is forest-based and deals with the internal nodes to progressively transmit n blocks, where n is not longer than the minimum number of either black or white blocks in the octree. This pointer-less octree is represented by its leaf nodes, where each leaf q is uniquely identified by the path leading from the root to q called *locational code*. This path is represented as a sequence of directional codes: 1 to 8 for each of the eight directions in the octree (UNW, UNE, USW, USE, DNW, DNE, DSW, DSE). The *locational code* for any node is an integer computed by a recursive function. Although this method is progressive, not all the approximations are a bounding volume of the original object.

The second method is a progressive solid simplification approach for 3D objects represented by binary space partition (BSP), presented by Huang and Wang [61]. It produces a LOD sequence with a decreasing number of nodes by using a combination of two techniques: (1) a volume bounded convex simplification that collapse parts with small volumes into a simple convex volume enclosing the volumetric cells on the input object, and (2) a plane collapse method which reduces the BSP tree depth ensuring that the input object is enclosed by the simplified BSP tree. In this way, from an initial object Γ^0, a progressive sequence of simplified BSP trees, Γ^i, can be computed, such that $\Omega(\Gamma^i) \subseteq \Omega(\Gamma^{i+1})$, $i = 0, \ldots, n-1$, where $\Omega(\Gamma^i)$ is the space occupied by Γ^i. As the simplified object is always a convex wrapping of the original, this approach is not suitable if the original object is not close to a convex shape.

Finally, the third method is based on a structure called bounding-planes octree (BP-O) proposed by Melero et al. [94], which is based on an octree where each node contains a set of planes taken from the input polygonal model. These planes form a convex bounding volume that includes completely the portion of the 3D object contained in the node. The nodes only contain the indices of the planes, so, all the planes are stored in an external data structure accessed by the octree. The same idea applies to the final geometry, stored at leaf nodes. To construct the BP-O, the polygons are classified in a 3D grid first, then, these polygons are assigned to each leaf node and all these nodes are grouped in order to have a minimum number of planes per node. After selecting the bounding planes at leafs, internal nodes are built by following the path from the root node to each leaf, computing their bounding planes in a bottom-up recursion.

One of the applications of BP-Octrees is the progressive transmission of 3D data though the network [93], in this case, planes contained in low levels of the BP-Octree are sent first, and so on until the leaf level. To save the generated data, it is distributed into three files, and as the data size of the original geometry is smaller than the required data size to transmit the full resolution contained in the BP-Octree, it is clear that the method does not compress the information. Besides, this approach is also not suitable if the original object is not close to a convex shape.

2.5 Conclusions

The purpose of this chapter was to give a brief overview of the state of art of the issues that conform the base of this thesis. Some conclusions have been derived from this review and they have oriented the objectives of this thesis:

- Scanned images with 3D acquisition devices must be represented with some representation model in order to manipulate them. Datasets used in this thesis are restricted to binary volumes and there are many models to represent them, where each one offers different capabilities and restrictions. The most common model is the voxel model, but several proposals as EVM and OUDB, represent binary volume datasets in a more compact way.

- Most of the methods to compute the pore-size distribution of porous samples rely on a previous computation of a skeleton (1D or 2D), which is a very time-consuming process.

- The study of other properties of materials, collectively called structural parameters, is of interest for researchers. Several methods exist to compute the connectivity mainly based on classical representation models. Since the measurement of a 3D roundness index is impractical, methods to compute sphericity and roundness work with 2D projections of the sample.

- Because of the size of the source models, sometimes it is desirable to compute a simplified model. Moreover, if the approximation is also a bounding volume, it can be used in many applications such as collision detection, volume of interest computation, among other tasks.

CHAPTER 3

An Improved Decomposition Model for OPP

3.1 Introduction

In this chapter, the Compact Union of Disjoint Boxes (CUDB) model for orthogonal pseudo-polyhedra (OPP) is presented. CUDB is a special kind of cell decomposition representation which performs a spatial partition along the OPP geometry in a non-hierarchical sweep-based way. CUDB improves OUDB (see Section 2.2.5) by reducing the number of boxes and preserving the adjacency information.

Structural parameters computation as well as the simplification approach presented in the next two chapters are based on EVM and CUDB. Therefore, in this chapter, algorithms for conversion between EVM and CUDB as well as basic CUDB algorithms such as CCL and collision (adjacency) detection, used in the mentioned methods, are also presented. We show that the aforementioned CUDB operations are more efficient than the corresponding OUDB-based ones, due to the fact that the number of elements to be analyzed is notoriously reduced.

3.2 Compact Union of Disjoint Boxes Model

Like OUDB, CUDB is also a union of disjoint boxes but a more compact one as several contiguous boxes are merged into one in several parts of the model. Let P be an OPP, to obtain the ABC-OUDB model, P is subdivided by planes perpendicular to the A-axis first, and then by planes perpendicular to the B-axis, at each cut C_i of P. Thus, every C_i splits all the geometry of P along the corresponding plane, and therefore some local regions of P, with which C_i actually has no relationship, are further unnecessarily divided. Figure 3.1 shows this situation for the YZX-OUDB of an OPP P, where some cuts force unnecessary divisions. For OUDB this constraint is mandatory to keep sorted the resulting boxes. However, in order to subdivide just the pieces of P related with the cut which induces the splitting, this constraint can be relaxed.

Formally, let P be an OPP. The $CUDB(P)$ can be obtained by merging boxes in several parts of the corresponding $OUDB(P)$. Then, this model is the set of boxes obtained according

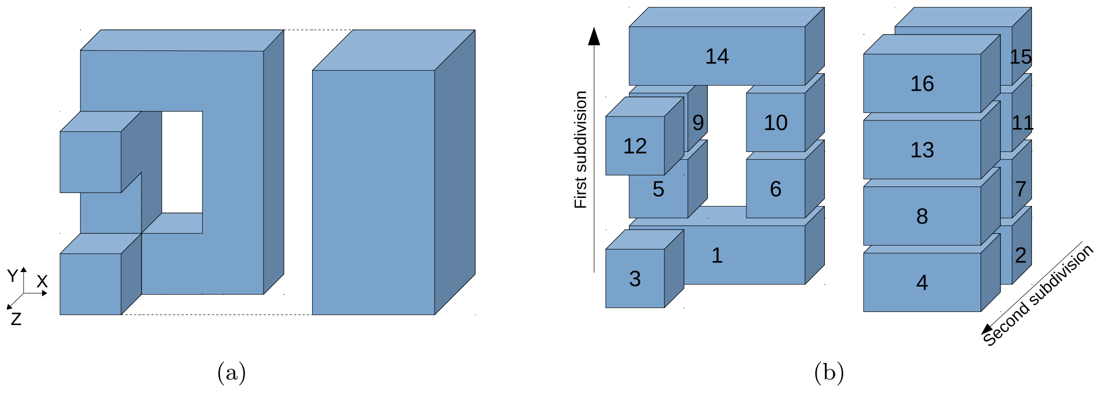


Figure 3.1: (a) An OPP. (b) YZX-OUDB with 16 boxes.

to the next properties:

1. Let β_1 and β_2 be two adjacent boxes of $OUDB(P)$ in B-direction, and let $\overline{\beta_1}^B$ and $\overline{\beta_2}^B$ be their projections respectively onto the plane perpendicular to the B-axis, then β_1 and β_2 can be merged as a single box if $\overline{\beta_1}^B = \overline{\beta_2}^B$.

2. Let β_1 and β_2 be two adjacent boxes of $OUDB(P)$ in A-direction, and let $\overline{\beta_1}^A$ and $\overline{\beta_2}^A$ be their projections respectively onto the plane perpendicular to the A-axis, then β_1 and β_2 can be merged as a single box if $\overline{\beta_1}^A = \overline{\beta_2}^A$. Note that A-direction in this property is different of B-direction of the first property.

The first property merges all unnecessary subdivisions along B-axis. Following with the previous example, Figure 3.2(a) shows that the pairs of boxes (2, 4), (7, 8), (9, 12), (11, 13) and (15, 16) depicted in Figure 3.1(b) can be merged applying this property for the Z-axis.

The second property merges the remaining unnecessary subdivisions along A-axis. Following with the same example, the resulting model depicted in Figure 3.2(b) shows that applying this

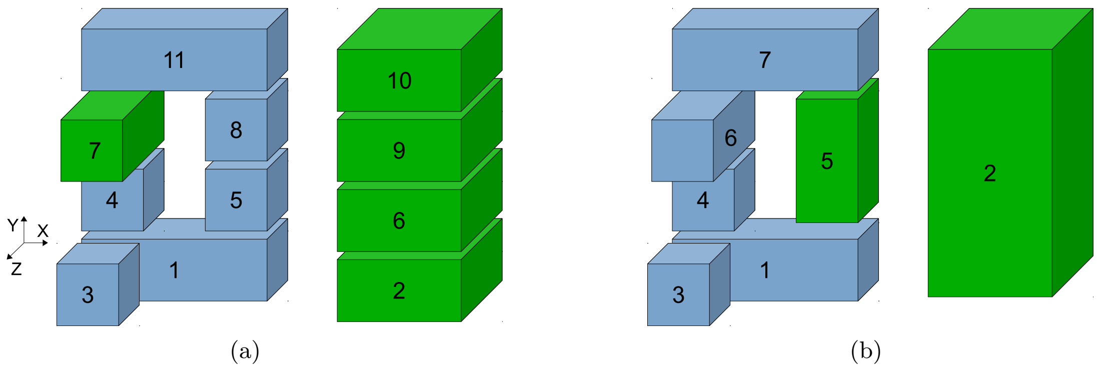


Figure 3.2: (a) Result after first merging in Z-direction. (b) Resulting YZX-CUDB with 7 boxes after merging in Y-direction.

second property for the Y-axis, not only the pairs of boxes (5, 8), but also the set of boxes (2, 6, 9, 10) depicted in Figure 3.2(a) can be merged.

Then, the CUDB-representation of an OPP P, is the set of disjoint boxes of the corresponding OUDB, conveniently reduced by applying the two previous merging properties. Let β_i be a box in $CUDB(P)$:

$$P = \bigcup_{i=1}^{n_b} \beta_i(P) \tag{3.1}$$

where n_b is the number of boxes, which is less or equal than the number of boxes in $OUDB(P)$.

Like OUDB, there are six different ABC-CUDB models for a given OPP, which correspond to each of the ABC-sorted EVM, and the number of obtained boxes depends on the ABC-sorting of the original EVM (see Figure 2.7) but we cannot know it a priori from EVM. Boxes in CUDB are sorted according to its coordinate A, then to coordinate B, and finally to coordinate C of its lower bound.

Although the implicit order among boxes in OUDB that defines their adjacency is lost, preserving the adjacency information in the CUDB model is easy with a tiny storage effort. Each box has neighboring boxes in only two orthogonal directions: A and B-direction, and for each one there are two opposite senses, so, four arrays of pointers to the neighboring boxes (two for each direction) are enough to preserve the adjacency information that is required for future operations. We define these arrays as A-backward neighbors (ABN), A-forward neighbors (AFN), B-backward neighbors (BBN) and B-forward neighbors (BFN).

To obtain the CUDB-representation it is not necessary to compute the OUDB model. CUDB can be computed directly from EVM because merging of boxes can be performed on the fly.

CUDB has been implemented as an object with a set of properties and methods. Next, we describe the CUDB object and the algorithms for conversion to and from EVM, for CCL and for collision detection.

3.2.1 CUDB structure

Each box in CUDB is an object (abstract data type) with the following primitive properties and methods. Let β be a box object and $V0$ and $V1$ be two objects of a predefined point3D data type:

Box properties:

- point3D $V0$, $V1$: point3D objects that represents the two diagonally opposed vertices of β, the ones with lowest and highest coordinate values.
- vector ABN, AFN, BBN, BFN: Vectors of pointers to the neighbors of β.
- integer *label*: The label of β.

Box methods:

- $\beta.createBox$(point3D $V0$, point3D $V1$): Creates a new box β with vertices $V0$ and $V1$. L is set to 0 (undefined) and vectors ABN, AFN, BBN and BFN to $\emptyset$.
- $\beta.getLabel()$: Returns the label of β.
- $\beta.setLabel$(integer $label$): Sets the label of β as $label$.

CUDB is an object with the following primitive properties and methods. Let Q be a CUDB object, and dimType and sortingType two predefined enumerated data types:

CUDB properties:

- boolean $allowNonManifolds$: Flag that indicates if the boxes adjacency allows non-manifold configurations.
- vector $boxes$: Vector containing the ordered boxes of Q.
- dimType dim: Dimension of Q. dimType={0D, 1D, 2D, 3D}
- integer $nBoxes$: Number of boxes in Q.
- sortingType $sort$: Sorting of Q. sortingType={XYZ, XZY, YXZ, YZX, ZXY, ZYX}

CUDB methods:

- $Q.createCUDB$(dimType dim, sortingType $sort$, boolean anm): Creates an empty CUDB object Q of dimension dim and sorting $sort$, and sets flag $allowNonManifolds$ as anm.
- $Q.getBox$(integer id): Returns the box at position id in the vector $boxes$.
- $Q.getDimension()$: Returns the dimension of Q.
- $Q.getNBoxes()$: Returns the number of boxes of Q.
- $Q.getNextBox$(Box β): Returns the next box to β in the vector $boxes$.
- $Q.getSorting()$: Returns the sorting of Q.
- $Q.insertBox$(Box β): Inserts the box β at the end of the vector $Q.boxes$, after its current last element.

These properties and methods are often referenced in the algorithms presented in this chapter and in the following ones.

3.2.2 EVM to CUDB Conversion

This process follows the same strategy that the process to compute OUDB. Cuts of the EVM-represented object are obtained sequentially, and sections are computed from them. When this process is performed in 2D, the corresponding sections result in the boxes of the OUDB model. For the EVM to CUDB conversion method, the same set of boxes is computed on the fly and a box is stored in the CUDB model if it is not possible to merge it with previously computed boxes applying the aforementioned Properties 1 and 2.

For a given box, the set of previous boxes that have to be considered for merging with it are those boxes belonging to the previous A-slice, which can be adjacent in A-direction, and those boxes belonging to the previous B-slice, which can be adjacent in B-direction. To facilitate this process, temporary lists of box pointers of the current and previous slices are maintained. The corresponding algorithm is detailed next.

As most of the algorithms dealing with EVM, the corresponding conversion algorithm is also recursive over the dimension. The main function $EVMtoCUDB$ (Algorithm 1) receives an EVM-represented OPP P and a flag to indicate if the adjacency relationship among boxes allows non-manifold configurations, and returns the CUDB-represented OPP Q containing the neighborhood information. This function initializes the temporary lists of box pointers *prevBBoxes*, *currentBBoxes*, *prevABoxes* and *currentABoxes*, which are defined as global variables throughout the whole conversion process, and starts the recursion by calling the function $processEVMtoCUDB()$ (Algorithm 2) with the original object P.

In function $processEVMtoCUDB()$, when dimension is 3D, the object is split at each cut in A-direction obtaining a set of 3D A-slices, $\zeta_A = \{\zeta_A^1, \zeta_A^2, \ldots, \zeta_A^{n_A}\}$, where n_A is the number of A-slices. Then the algorithm applies recursively to the 2D section representing each slice, which is split at every internal cut in B-direction obtaining a set of 2D B-slices, $\zeta_B = \{\zeta_B^1, \zeta_B^2, \ldots, \zeta_B^{n_B}\}$, where n_B is the number of B-slices represented by their 1D sections, which are composed by a set of collinear brinks in C-direction. Each of these brinks defines a box. Then, each 2D slice ζ_B^i defines one o more boxes, and each 3D slice ζ_A^j contains all the boxes defined in its 2D slices.

In the base case, when dimension is 1D, each brink in the current slice ζ_B^i results in a box, which is inserted into *currentBboxes*. Boxes in a 2D slice ζ_B^i can be merged with boxes in the previous slice ζ_B^{i-1}, then, in the backtracking step of the recursion when dimension is 2D,

Algorithm 1: EVMtoCUDB

```
Input  : $P$;                                              /* EVM-represented OPP /*
Input  : $allowNonManifolds$;                                               /* Flag /*
Output: $Q$;                                              /* CUDB-represented OPP /*
$dim \leftarrow P.getDimension()$;
$Q.createCUDB(dim, P.getSorting())$;
$Q.allowNonManifolds \leftarrow allowNonManifolds$;
$prevBBoxes \leftarrow \emptyset$; $currentBBoxes \leftarrow \emptyset$;
$prevABoxes \leftarrow \emptyset$; $currentABoxes \leftarrow \emptyset$;
$processEVMtoCUDB(P, Q, dim, \emptyset, \emptyset)$;                           /* First call /*
```

Algorithm 2: processEVMtoCUDB

```
Input  : $P$;                                          /* EVM-represented OPP /*
Input  : Output : $Q$;                                   // CUDB-represented $Q$
Input  : $dim$;                                        /* Dimension parameter /*
Input  : $V0$;                                        /* Lower vertex for boxes /*
Input  : $V1$;                                        /* Upper vertex for boxes /*
if $dim$ =$1D$ then
    for all brink $br \in P$ do
        $V0.C, V1.C \leftarrow br.readBrink()$;
        $\beta.createBox(V0, V1)$;  Add $\beta$ to $currentBBoxes$;
    end for
else                                                      /* $dim$ =2D or 3D /*
    $Sec \leftarrow \emptyset$;  $Cut, coordIni \leftarrow P.getNextCut()$;
    $Sec \leftarrow Sec \otimes Cut$;  $Cut, coordFin \leftarrow P.getNextCut()$;
    while $Cut \neq \emptyset$ do
        if $dim$ =$3D$ then
            $V0.A \leftarrow V1.A$;  $V1.A \leftarrow coordFin$;
        else                                                  /* $dim$ =2D /*
            $V0.B \leftarrow V1.B$;  $V1.B \leftarrow coordFin$
        end if
        $processEVMtoCUDB(Sec, Q, dim - 1, V0, V1)$;
        if $dim$ =$3D$ then
            $mergeA()$;
        else                                                  /* $dim$ =2D /*
            $mergeB()$;
        end if
        $Sec \leftarrow Sec \otimes Cut$;  $Cut, coordFin \leftarrow P.getNextCut()$;
    end while
end if
```

function $mergeB()$ (Algorithm 3) is called, which compares all boxes β_1 in *currentBboxes* with all boxes β_2 in ζ_B^{i-1} (stored in *prevBBoxes*) for merging. In this process, when merging property 1 is accomplished, $\beta_2 = \beta_2 \cup \beta_1$, and it is inserted into a list called *activeBoxes*. Otherwise, β_1 is inserted into *currentABoxes* and *activeBoxes*. When the process finishes, list *activeBoxes* becomes *prevBBoxes* in order to be compared with boxes in ζ_B^{i+1} in the next call.

Similar to the merging case in B-direction, boxes in a 3D slice ζ_A^j can be merged with boxes in the previous slice ζ_A^{j-1}. Once all the boxes of the current slice ζ_A^j have been computed and conveniently merged in B-direction, they are in *currentAboxes*. Then, in the backtracking step of the recursion when dimension is 3D, function $mergeA()$ (Algorithm 4) is called, which compares all boxes β_1 in *currentAboxes* with all boxes β_2 in ζ_A^{j-1} (stored in *previousAboxes*) for merging. The steps in this function are quite similar to those in function $mergeB()$, but in this case, the merged boxes are finally inserted into the CUDB model Q.

Note that, the adjacency information of boxes (ABN, AFN, BBN, BFN) is computed on the fly when tests for merging are performed.

Algorithm 3: mergeB

```
$activeBoxes \leftarrow \emptyset$;
for all $\beta_1 \in currentBBoxes$ do
    for all $\beta_2 \in prevBBoxes$ do
        if $\overline{\beta_1}^B = \overline{\beta_2}^B$ then                    /* Merging property 1 /*
            $\beta_2 \leftarrow \beta_1 \cup \beta_2$;  Add $\beta_2$ to $activeBoxes$;
            break;
        else if $\overline{\beta_1}^B \cap \overline{\beta_2}^B \neq \emptyset$ then   /* According to flag $allowNonManifolds$ /*
            Add $\beta_2$ to $\beta_1.BBN$;
        end if
    end for
    if $\beta_1$ was not merged then
        Add $\beta_1$ to $currentABoxes$;  Add $\beta_1$ to $activeBoxes$;
        Add $\beta_1$ as BFN for each box in $\beta_1$.BBN;
    end if
end for
$prevBBoxes \leftarrow activeBoxes$;  $currentBBoxes \leftarrow \emptyset$;
```

Algorithm 4: mergeA

```
$activeBoxes \leftarrow \emptyset$;
for all $\beta_1 \in currentABoxes$ do
    for all $\beta_2 \in prevABoxes$ do
        if $\overline{\beta_1}^A = \overline{\beta_2}^A$ then                    /* Merging property 2 /*
            $\beta_2.BBN = \beta_2.BBN \cup \beta_1.BBN$;  $\beta_2.BFN = \beta_2.BFN \cup \beta_1.BFN$;
            $\beta_2.ABN = \beta_2.ABN \cup \beta_1.ABN$;  $\beta_2.AFN = \beta_2.AFN \cup \beta_1.AFN$;
            $\beta_2 \leftarrow \beta_1 \cup \beta_2$;  Add $\beta_2$ to $activeBoxes$;
            break;
        else if $\overline{\beta_1}^A \cap \overline{\beta_2}^A \neq \emptyset$ then   /* According to flag $allowNonManifolds$ /*
            Add $\beta_2$ to $\beta_1.ABN$;
        end if
    end for
    if $\beta_1$ was not merged then
        $Q.insertBox(\beta_1)$;  Add $\beta_1$ to $activeBoxes$;
        Add $\beta_1$ as AFN for each box in $\beta_1$.ABN;
    end if
end for
$prevABoxes \leftarrow activeBoxes$;  $currentABoxes \leftarrow \emptyset$;
```

3.2.3 CUDB to EVM Conversion

As in OUDB, all of the boxes in CUDB are disjoint. Therefore, according to EVM Property 5 (see Section 2.2.4), a simple XOR operation of all the EVM-represented boxes is necessary to get the EVM-represented object. That is, let β_i be a box in the CUDB-represented OPP P.

$$EVM(P) = \bigotimes_{i=1}^{n_b} EVM(\beta_i) \tag{3.2}$$

where $EVM(\beta_i)$ is the EVM-representation of β_i and n_b is the number of boxes in CUDB.

3.2.4 Area and Volume Computation

Computing the volume (3D) or area (2D) is straightforward by doing a traversal of the boxes, and making a summation of each volume or area depending on the dimension.

$$Volume(P) = \sum_{i=1}^{n_b} Volume(\beta_i) \tag{3.3}$$

3.2.5 Connected Component Labeling

Connected Component Labeling (CCL) is a very important operation for managing volume datasets where multiple disconnected components that compose the volume need to be identified. Traditional voxel-based methods have been widely used [128]. OUDB has been proved to be efficient for CCL [124, 125]. With regard to semi-boundary representations, it has been concluded that CCL is better in OUDB than in semi-boundary representations when the number of boxes in the OUDB is less than the number of boundary voxels, which generally occurs.

The typical implementation of the aforementioned approaches is based in the classical two-pass strategy [128]: the *labeling pass* and the *renumbering pass*. In short, in the *labeling pass* all elements are scanned and labeled according to their already labeled neighbors. Some labeling ambiguities can be produced in this step which are properly registered in a set of equivalence classes. Then, the renumbering pass solves these ambiguities and the elements are relabeled.

In the OUDB-CCL process [124], the traversal of the boxes is performed orderly, so, checking the neighborhood of the current box involves those boxes in the immediate previous B-slice and those boxes in the immediate previous A-slice. An improvement of the OUDB-based CCL has been already proposed [11], where the so-called OUDB-extended is computed, which allows jumping directly to the required box that needs to be tested, instead of querying and skipping several intermediate boxes. However, the main drawback of previous approaches [11, 124] is the large size of the equivalence table, because they need one entry per each new detected label.

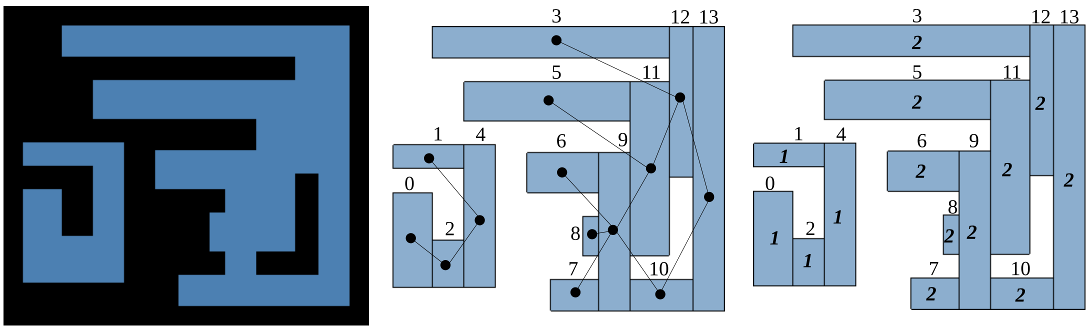
The same two-pass strategy for CCL can be applied in CUDB. However, as CUDB contains the boxes neighborhood information, it can bee seen as an undirected graph. Thus, the CUDB-CCL process is based on the detection of connected components in graph theory.

Let $G = (V, E)$ be an undirected graph without self loops, with V being a set of vertices (the CUDB boxes) and E a set of edges defined by the neighborhood information (ABN, AFN, BBN and BFN). A connected component in G is a maximal subgraph $S = (V^S, E^S)$ in which for any two vertices $v, u \in V^S$ there exists an undirected path in G with v as start and u as end vertex [145]. A maximal subgraph means that for any additional vertex $w \in (V \backslash V^S)$ there is no path from any $v \in V^S$ to w.

Thus, the CUDB-CCL process has linear complexity, in terms of the sum of the numbers of vertices and edges of the graph, using either depth-first search or breadth-first search [60]. In either case, a search that begins at some box β, will find the entire connected component containing β. To detect all the connected components, a traversal of the boxes is performed, starting a new breadth-first search or depth-first search whenever a box that has not been already labeled is detected. Algorithm 5 details the steps for the CUDB-CCL process using a breadth-first search strategy. Figure 3.3 depicts the CUDB-CCL process for a 2D example, where the evolution of the boxes queue used by the algorithm is shown.

Algorithm 5: CCL

```
Input  : $Q$;                                            /* CUDB-represented OPP /*
Output: $Q$;                                     /* Labeled CUDB-represented OPP /*
Output: $cc$;                                   /* Number of connected components /*
$currentLabel \leftarrow 1$;
$\Delta = \emptyset$ ;                                         /* Queue of box pointers /*
for all $\beta \in Q$ do
    if $\beta.getLabel() = \emptyset$ then
        $\beta.setLabel(currentLabel)$;  Add $\beta$ to $\Delta$;
        while $\Delta \neq \emptyset$ do
            $\delta = \Delta.front()$;  $\Delta.pop()$;     /* Get and remove the next box in $\Delta$ /*
            for all $\gamma \in (\delta.ABN \cup \delta.AFN \cup \delta.BBN \cup \delta.BFN)$ do
                if $\gamma.getLabel() = \emptyset$ then
                    $\gamma.setLabel(currentLabel)$;  $\Delta.push(\gamma)$;
                end if
            end for
        end while
        $currentLabel = currentLabel + 1$;
    end if
end for
$cc \leftarrow currentLabel - 1$;
```



Original object **CUDB representation (edges defined by the neighborhood)** **CUDB after labeling (in bold the label number)**

Evolution of the boxes queue:

Label =1: {0} → {2} → {4} → {1} → {}

Label =2: {3} → {12} → {11, 13} → {13, 5, 9} → {5, 9,10} → { 9,10} → {10, 6, 7, 8} → { 6, 7, 8} → {7, 8} → {8} → {}

Figure 3.3: 2D example of CUDB-CCL.

3.2.6 Exact Collision Detection

Collision detection is an important characteristic in representation models. Sometimes we want to determine if two or more objects collide or are adjacent. In collision detection [64] when exact accuracy is not required, typical bounding volumes like AABB, spheres, oriented polytopes or hybrid bounding volumes are used. However, when accuracy is important, a thorough analysis of the contact between the involved objects needs to be done.

A straightforward solution in CUDB is to iteratively compare each of the boxes in an object with all of the boxes in the other objects (brute force). Nevertheless, taking advantage of the implicit order of the boxes in the CUDB model, unnecessary analysis can be avoided. A technique to detect pairs of colliding objects from a collection of n CUDB-represented objects is presented next. For this method it is mandatory that the CUDB-represented objects have the same ABC-ordering; otherwise, a preprocesing must be performed first in order to set the same ordering.

In complex scenes there might be several objects interacting. In such cases, an early detection phase can be applied to discard collisions between objects which are not close enough using some bounding volume. Sweep and prune algorithms [27, 44] sort the objects according to the lower and upper bounds of their bounding volumes, and when a pair of objects are very close, it is tested to exact collision. In the presented method, a discarding of those objects whose AABB do not collide is performed first. Then, as boxes in CUDB are ABC-sorted, all the remaining *potentially colliding* objects can be tested jointly, instead of testing them in pairs.

Let $\Theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$ be a finite sequence of n CUDB-represented *potentially colliding* objects, and let $\Delta = \{\beta_1, \beta_1, \ldots, \beta_n\}$ be a set of box pointers, where each β_i points to a box in the object θ_i. Initially each β_i points to the first box of the corresponding object.

A collision detection between all of the boxes in Δ is performed first, followed by an iterative process. This process obtains the box β_{min} in Δ (β_i with the minimum ABC-position of its vertex $V0$) and updates it with the next box in the object θ_{min}. If there are no more boxes in θ_{min}, this object is marked as not active. Otherwise, β_{min} is compared for collision with all boxes $\beta_i \in \Delta, \forall i \neq min$ and with the subsequent neighboring boxes of each β_i, say β_t, until $\beta_t.V0$ has an A-coordinate greater than $\beta_{min}.V1$. Note that we do not need to compare B and C-coordinate. The main iterative process finishes when β_{min} cannot be defined, which means that all θ_i have been marked as not active. At the end, a set S, with object pairs (θ_i, θ_j) that collide or are adjacent has been defined.

Algorithms 6 and 7 detail the steps of this process. Function *getIndexMinBox()* returns the index min of the box in Δ with the minimum ABC-position of its vertex $V0$, such that θ_{min} is marked as active. If all θ_i are marked as no active, this function returns $\emptyset$. The worst case time-complexity of the CUDB-based exact collision detection is $O(n{\cdot}\, m{\cdot}\, M)$, where n is the number of objects, m the number of boxes of the object having the maximum number of boxes, and M the total number of boxes in the n objects. In any case it holds that $n \leq m \leq M$.

Algorithm 6: detectCollision

```
Input  : $\Theta$;                                    /* Set of CUDB-represented objects /*
Output: $S$;                                  /* Set of colliding object pairs ($\theta_i$, $\theta_j$) /*
$Act \leftarrow \emptyset$ ;                                /* Vector of flags for active $\theta_i$ /*
$\Delta \leftarrow \emptyset$ ;                                      /* Vector of box pointers /*
for all $\theta \in \Theta$ do
    $\beta \leftarrow \theta.getBox(0)$;  Add $\beta$ to $\Delta$;  Add true to $Act$;
end for
for all $\beta_i \in \Delta$ do                                        /* First collision test /*
    $testCollision(\Delta, i, S)$
end for
$min \leftarrow getIndexMinBox(\Delta, Act)$;
while $min \neq \emptyset$ do
    $\beta_{min} \leftarrow \theta_{min}.getNextBox(\beta_{min})$;
    if $\beta_{min} \neq \emptyset$ then $testCollision(\Delta, min, S)$;
    else $Act[min]$ =false; end if;                       /* Mark $\theta_{min}$ as not active /*
    $min \leftarrow getIndexMinBox(\Delta, Act)$;
end while
```

Algorithm 7: testCollision

```
Input  : $\Delta$;                                              /* Vector of box pointers /*
Input  : $i$;                                         /* Index of the test box in $\Delta$ /*
Input  : Output : $S$ ;                       /* Set of colliding object pairs ($\theta_i$, $\theta_j$) /*
for all $\beta_j \neq \beta_i \in \Delta$ do
    $\beta_t \leftarrow \beta_j$;
    while $\beta_t \neq \emptyset$ do
        if $\beta_t.V0.A > \beta_i.V1.A$ then break; end if;
        if $\beta_i \cap \beta_t \neq \emptyset$ then Add pair $(\theta_i, \theta_j)$ to $S$;  break; end if;
        $\beta_t \leftarrow \theta_j.getNextBox(\beta_t)$;
    end while
end for
```

3.3 CUDB Performance

CUDB has been compared with OUDB in number of elements and computation time for conversion to and from EVM and CCL (using OUDB-extended version [11]). The test datasets consists of 15 objects (see Figure 3.4). All datasets come from public volume repositories, where from (h) to (o) are real volume models coming from CT or MRI scanners. The corresponding programs have been written in C++ and tested on a PC Intel®Core 2 Duo CPU E6600@2.40GHz with 3.2 GB RAM and running Linux.

Table 3.1 shows the attributes of test datasets. Table 3.2 shows a comparison respect the number of boxes in OUDB and CUDB where two possible orderings are considered (XYZ and ZYX). Note that, although the number of boxes depends on the ABC-sorting of the original EVM, CUDB produces less elements than EVM and OUDB in all cases, in some of them less than 10% of elements. For instance, the XYZ-CUDB representation of the Lines dataset has

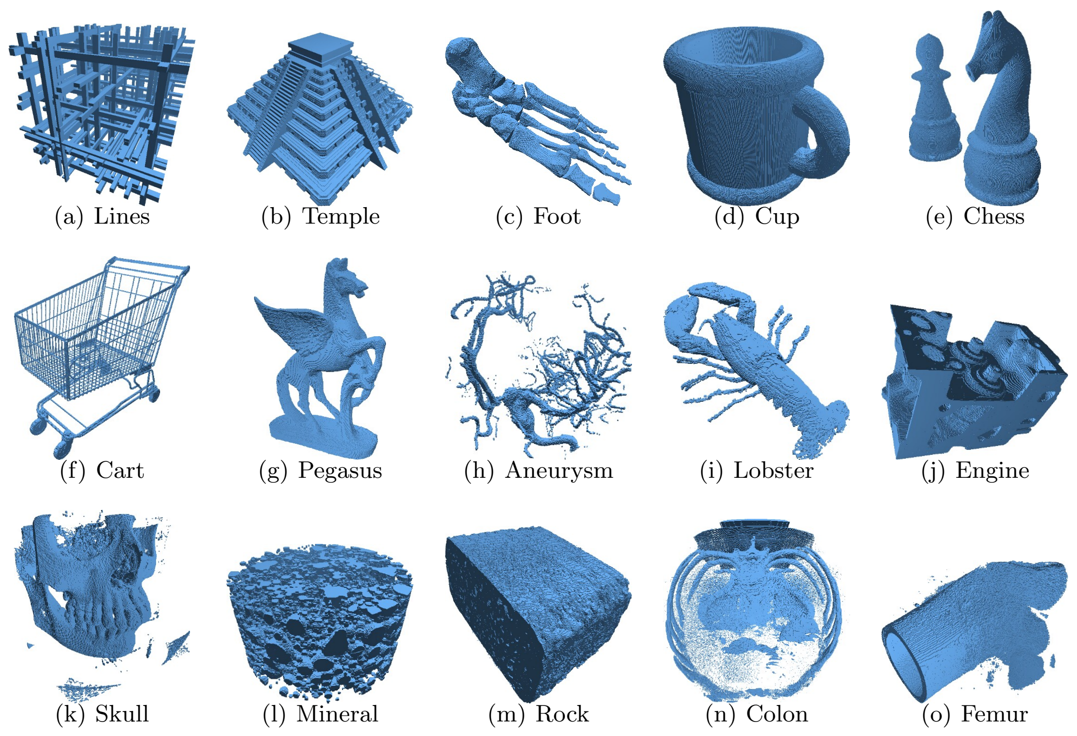


Figure 3.4: Rendered images of the test datasets.

Table 3.1: Attributes of the test datasets. For each dataset: size in voxels, number of extreme vertices ($|EV|$) and number of connected components ($|CC|$) for 26-adjacency.

Dataset	size	$\|EV\|$	$\|CC\|$
Lines	500×500×500	4356	1
Temple	925×1000×472	73902	99
Foot	183×512×185	140012	6
Cup	401×401×512	215050	1
Chess	511×246×480	287360	2
Cart	585×979×1000	502986	5
Pegasus	598×800×574	709960	1
Aneurysm	213×215×240	50318	406
Lobster	244×239×49	74724	53
Engine	139×197×108	101114	9
Skull	256×256×256	506454	1624
Mineral	376×375×206	833002	724
Rock	240×406×267	1317106	1336
Colon	512×492×426	2142304	54829
Femur	463×492×628	3584724	22714

only 3.84% of boxes of the corresponding XYZ-OUDB representation (see Figure 3.5).

Table 3.3 shows the performance of both OUDB and CUDB. Although the conversion from EVM to CUDB is a little slower than EVM to OUDB due to the extra effort to merge the

Table 3.2: OUDB and CUDB size comparison. For each dataset: number of boxes in OUDB ($|OUDB|$), CUDB ($|CUDB|$) and ratio considering the XYZ and ZYX-ordering.

Dataset	XYZ-ordering			ZYX-ordering		
	$\|OUDB\|$	$\|CUDB\|$	$\%\frac{\|CUDB\|}{\|OUDB\|}$	$\|OUDB\|$	$\|CUDB\|$	$\%\frac{\|CUDB\|}{\|OUDB\|}$
Lines	24851	954	3.84	20081	963	4.80
Temple	260279	21084	8.10	77701	18456	23.75
Foot	42671	35498	83.19	50868	34778	68.37
Cup	236642	60429	25.54	64093	46342	72.30
Chess	92919	66235	71.28	96373	57258	59.41
Cart	256370	100358	39.15	289839	113996	39.33
Pegasus	289827	191747	66.16	189078	160367	84.82
Aneurysm	12825	10705	83.47	13560	11043	81.44
Lobster	27307	19322	70.76	22226	17189	77.34
Engine	47143	25524	54.14	52229	26371	50.49
Skull	154304	114563	74.24	169920	120459	70.89
Mineral	489585	232008	47.39	582624	267368	45.89
Rock	420795	331491	78.78	428603	337005	78.63
Colon	653717	473649	72.45	542202	440360	81.22
Femur	1172072	838585	71.55	1125560	838641	74.51

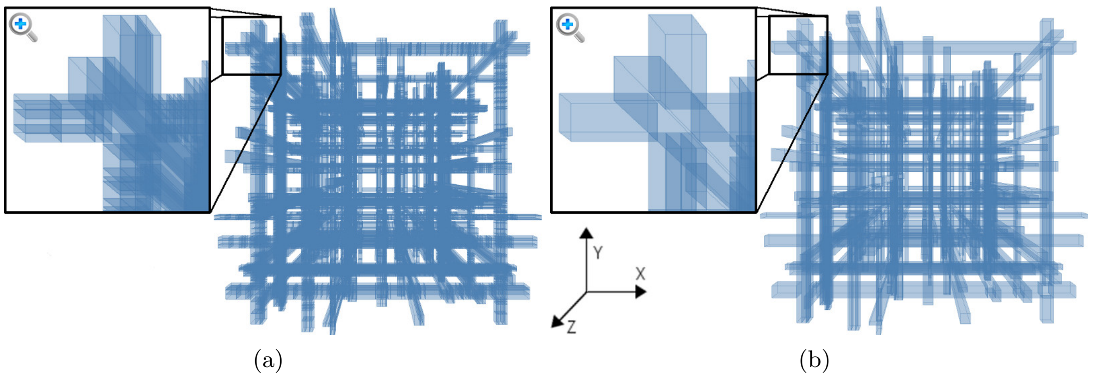


Figure 3.5: Lines dataset. (a) XYZ-OUDB representation with 24851 boxes. (b) XYZ-CUDB representation with 954 boxes.

boxes, the inverse conversion is faster due to the less number of elements, and regarding the CCL process, it is much faster in CUDB. Moreover, when computing the number of connected components starting from the EVM model, CUDB is more efficient than OUDB (see the last columns in this table).

In order to show the performance of the exact collision detection in CUDB, three scenes are presented. The first one (Figure 3.6) consists of seven datasets, where the size of each is around 128^3. The second scene (Figure 3.7) consists of 200 objects of the Star dataset randomly placed in a volume of 600^3 voxels. The third scene (Figure 3.8) consists of 2 objects of the Cart dataset that have interlaced parts but do not collide.

Statistics of the collision detection test are shown in Table 3.4. Note that, although scene 3

Table 3.3: OUDB and CUDB run time comparison in milliseconds. For each dataset the time for: EVM to OUDB ($E \to O$) and OUDB to EVM ($O \to E$) conversion, OUDB-CCL (O_{CCL}), EVM to CUDB ($E \to C$) and CUDB to EVM ($C \to E$) conversion and CUDB-CCL (C_{CCL}). The last columns represent $t_O = E \to O + O_{CCL}$, $t_C = E \to C + C_{CCL}$, and the ratio between t_O and t_C.

Dataset	OUDB			CUDB			t_O	t_C	$\%\frac{t_C}{t_O}$
	E→O	O_{CCL}	O→E	E→C	C_{CCL}	C→E			
Lines	51	31	81	51	1	4	82	52	63.41
Temple	713	831	704	799	4	87	1544	803	52.01
Foot	244	106	149	254	8	142	350	262	74.86
Cup	761	549	748	830	13	251	1310	843	64.35
Chess	357	117	319	439	15	269	474	454	95.78
Cart	862	638	883	978	33	491	1500	1011	67.40
Pegasus	1286	948	994	1511	66	821	2234	1577	70.59
Aneurysm	95	12	67	96	1	62	107	97	90.65
Lobster	142	35	132	151	4	83	177	155	87.57
Engine	165	79	177	205	5	113	244	210	86.07
Skull	628	260	588	735	34	513	888	769	86.60
Mineral	1323	1298	1646	1591	76	1033	2621	1667	63.60
Rock	1713	1330	1648	2055	102	1504	3043	2157	70.88
Colon	2766	1589	2684	3266	141	2295	4355	3407	78.23
Femur	4835	3947	5049	5688	282	4189	8782	5970	67.98

Table 3.4: Statistics of the collision scenes. For each scene: total number of boxes ($|boxes|$) in the scene, number of detected collisions (i.e., number of object pairs $|pairs|$) and time to detect the collisions in milliseconds.

Scene	$\|boxes\|$	$\|pairs\|$	Time (ms)
Scene 1	21963	4	17
Scene 2	256000	72	169
Scene 3	202101	0	2085

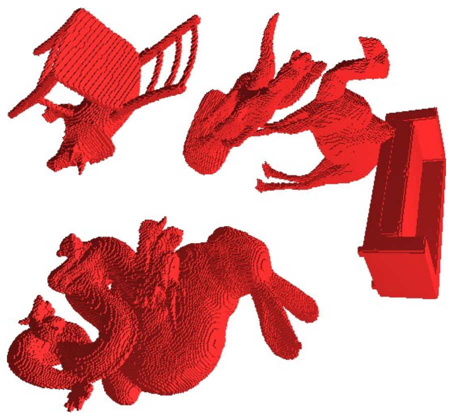


Figure 3.6: Collision scene 1. 7 datasets: Bunny (5895 boxes), Camel (2856 boxes), Chair (1693 boxes), Dragon (4853 boxes), Pegasus (4861 boxes), Sofa (141 boxes), Triceratops (1664 boxes). All objects collide with some other object.

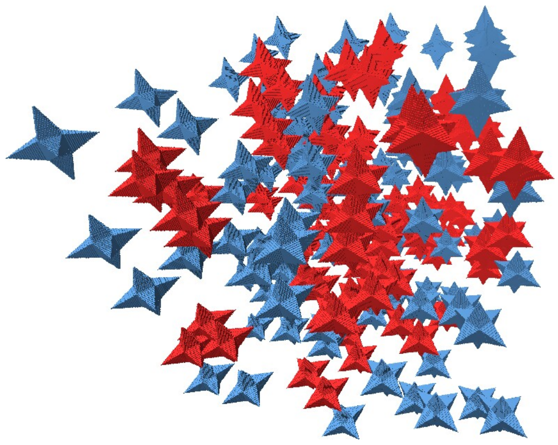


Figure 3.7: Collision scene 2. 200 objects of the Star dataset. Each object has 1280 boxes in its XYZ-CUDB representation. In red the objects that collide.

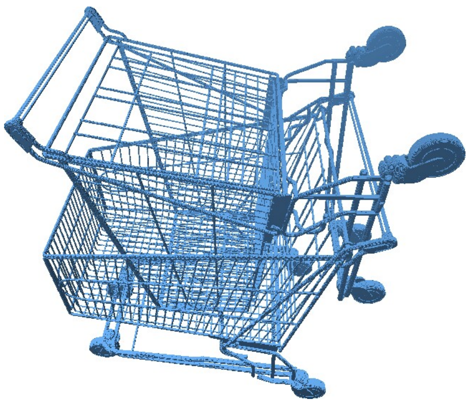


Figure 3.8: Collision scene 3. 2 objects of the Cart dataset. The original Cart has 100358 boxes, the rotated one 101743 boxes. Both in its XYZ-CUDB. The objects actually do not collide.

has less boxes than scene 2, the required time for collision detection is bigger. This is because there is any collision, which implies that there is not any early discarding and all boxes must be evaluated.

3.4 Conclusions

This chapter has presented a new decomposition model for OPP, CUDB, which is an improved version of OUDB. Algorithms for conversion to and from EVM, for CCL and exact collision (adjacency) detection have also been presented. Experimental results show that CUDB is smaller in number of elements, and so in storage size, and has a better performance for CCL than its improved version, OUDB-extended. Although the presented exact collision detection algorithm is CPU-based, it is efficient when exact collision detection is required directly on CUDB models. CUDB model has been satisfactorily applied in the computation of some structural parameters in the next chapter.

CHAPTER 4

Structural Parameters Computation

4.1 Introduction

This chapter presents several methods to compute structural parameters of binary volume datasets using CUDB and EVM as representation models. The computed parameters are:

Pore-size distribution. Section 4.2 presents a new approach to simulate MIP.

Connectivity. Section 4.3 presents a CUDB-based method to compute the Euler characteristic (χ) and the genus.

Sphericity and roundness. Section 2.3.5 presents alternative methods to the compute sphericity and roundness indices.

4.2 CUDB-based Virtual Porosimeter

In this section, a new approach to simulate MIP is presented. The novelty of this method is that, unlike other MIP simulation approaches, it does not require prior computation of the model's skeleton. This approach simulates mercury intrusion by detecting throats with a geometric method and then performing a CCL process. The size of the detected throats is related to the diameter corresponding to each increasing level of pressure.

In most of the reported bibliography, the operations to study the pore space are performed directly on the classical voxel model. However, in the field of volume analysis and visualization, several alternative models have been devised for specific purposes. Hierarchical models such as octrees or kd-trees are not used in the presented method because it does not need a hierarchy and because the CCL process is better suited to a sweep-based decomposition than to a tree-based one. Sweep-based decomposition models such as OUDB or CUDB are best suited and as CUDB has a better performance than OUDB, the presented method is based on this model.

4.2.1 Method Overview

Unlike traditional approaches, which require the skeleton and the voxelization of the pore space, the presented method uses the CUDB-encoding of the pore space. It consists of three different ABC-sorted CUDB with different A and C-coordinate (as explained later). Let us consider the XYZ, YZX and ZXY-sorting. Then, for each diameter D_i corresponding to successive applied pressures ρ_i, $\rho_i < \rho_{i+1}$, $i = 1, \ldots, n-1$, the invaded region R_i is computed by an iterative process. Each pressure ρ_i is related to a representative diameter D_i by the Washburn equation (see Equation 2.10).

For each iteration, three main steps are applied to the CUDB-represented pore space in order to simulate the mercury intrusion:

1. First, all pore regions smaller than the current diameter D_i are discarded (see Sec. 4.2.2).
2. Second, in order to prevent improper fluid flow along the pore space, the throat detection step determines all the transitions, smaller than the current diameter, between adjacent pore regions (see Section 4.2.3 and 4.2.4).
3. Finally, the third step simulates the mercury intrusion, for the current diameter, by labeling boxes as invaded or not invaded, constrained by the marked narrow throats and mercury entry points (see Section 4.2.5).

4.2.2 Discarding Process

The first step of the simulation consists in the removal of all pore regions smaller than the current diameter D_i. Granulometry-based approaches apply mathematical openings to eliminate all the regions smaller than a given structuring element, while keeping the larger ones almost unchanged. The opening of a set S by a structuring element E can be expressed in terms of the morphological operations erosion ($\ominus$) and dilation ($\oplus$) as:

$$S \odot E = (S \ominus E) \oplus E$$

In these approaches, the pore space is represented by voxels on a cubic lattice and the structuring element E is naturally given by a digital representation, as well. In the CUDB-based approach, an opening-like method is used to discard small regions of the CUDB-represented pore space that does not require explicitly performing both the erosion and dilation operations. Moreover, it uses a CUDB-box with edge length D as structuring element.

As boxes of an ABC-sorted CUDB have no neighbors in the C-direction, the strategy is to scan three ABC-sorted CUDB encodings of the pore space with different C-coordinate. Then, the C-edge length of each box is tested to determine whether or not it would disappear after erosion. For each sorting if the C-edge length of the current box is smaller than D, it is discarded; otherwise, it is preserved.

The final result of this step is not exactly equivalent to the conventional opening operation because, in the CUDB-based approach, the opening is carried out independently in each orthogonal direction. Thus, erosion and dilation are not actually performed; however, all regions smaller than the structuring element are removed in a fraction of the total time of a conventional opening operation. Figure 4.1 illustrates the discarding process with a 2D example for greater clarity. The 2D sample shown in the figure corresponds to the cross section of a 3D synthetic sample.

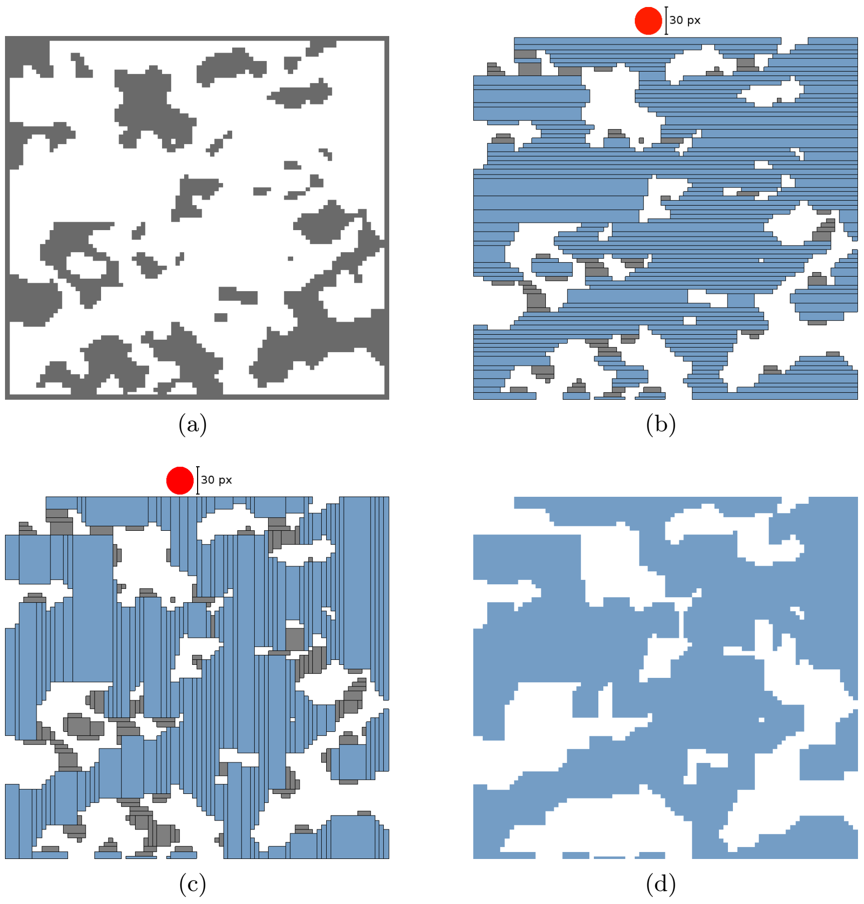


Figure 4.1: Discarding process for a 2D example. (a) Segmented pore space (in white here and blue for the other images). (b) YX-sorting: boxes with X-edge length smaller than a given diameter D are discarded (gray). (c) XY-sorting: boxes with Y-edge length smaller than D are discarded (gray). (d) Resulting pore region after two discarding steps.

4.2.3 Narrow Throats Detection

Once the discarding step has been completed, all regions smaller than the current diameter have been removed from the original pore space. However, there are still transitions between adjacent pore regions that are not defined by any edge of boxes, and, thus, will not have been detected in the discarding process. When these transitions are smaller than the current diameter, they are called narrow throats. Narrow throats prevent full mercury invasion of the whole pore space at the current intrusion pressure, so detecting them is mandatory to simulate the fluid flow correctly. Transitions larger than the diameter for the current intrusion pressure are ignored. Figure 4.2 illustrates the narrow throats for a given diameter represented by a ball. As can be seen, a fluid with the given diameter could enter the boxes, but it could not pass through the throats (in yellow).

Narrow throats can be orthogonal or oblique and all of them are shaped as rectangles. Due to the orthogonal nature of the CUDB model, there are two possible configurations of oblique throats: a single oblique throat, represented by a rectangle (see Figure 4.2(c)), and the general case, represented by three rectangles (see Figure 4.2(d)).

The orientation of rectangles representing orthogonal throats is such that two components of the normal vector of its supporting plane are zero, while for rectangles corresponding to oblique throats, one component is zero. Therefore, there are three possible single oblique throats, corresponding to the three main directions (see the small boxes in Figure 4.2(d)). In a general oblique throat, three rectangles are obtained, one in each main direction, and the throat is constructed from them.

In order to detect orthogonal and oblique narrow throats, the remaining pore space after the discarding process must be exhaustively scanned in the three main directions. Once all throats have been detected, they are represented with a 3D object that must be removed from the pore space to prevent the mercury invasion. This operation is performed using EVM due to its efficiency with Boolean operations.

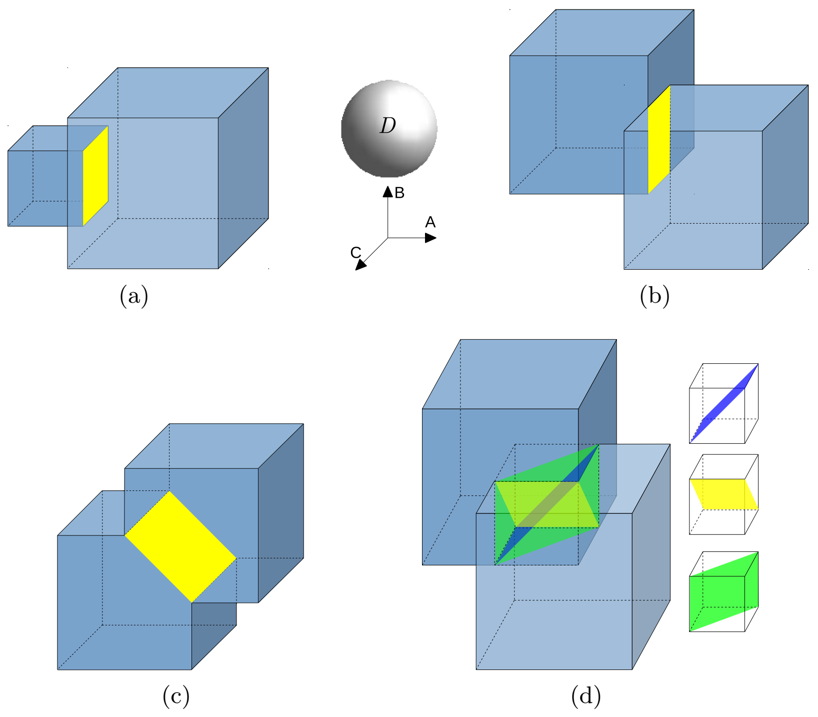


Figure 4.2: 3D narrow throats. (a) and (b) Orthogonal throats. (c) Single oblique throat. (d) Three oblique throats in the three main directions (blue, yellow and green).

Orthogonal Throats Detection

Boxes in CUDB have neighboring boxes in the A and B-direction, therefore, orthogonal throats can exist in any of these directions. As the first main axis that splits the model is A, orthogonal throats are detected in this direction. Thus, three ABC-sorted CUDB encodings of the pore space with different A-coordinate are required to detect all orthogonal throats.

Let β_i and β_j be two A-adjacent boxes, and $\overline{\beta_i}^A$ and $\overline{\beta_j}^A$ the open sets[1] of their projections respectively, over a plane perpendicular to the A-coordinate. There is an orthogonal throat between β_i and β_j if the next conditions are satisfied:

$$(\overline{\beta_i}^A \cap \overline{\beta_j}^A) \neq \emptyset \tag{4.1a}$$

$$length_C(\overline{\beta_i}^A \cap \overline{\beta_j}^A) < D \tag{4.1b}$$

where D is the diameter corresponding to the current intruded pressure and $length_C()$ returns the C-edge length of the rectangle formed by $\overline{\beta_i}^A \cap \overline{\beta_j}^A$. A $length_B$ condition is not considered because the detected throat could be part of a longest one in the B-direction, which does not occur in the C-direction as CUDB-boxes do not have neighbors in this direction. Condition 4.1a can be easily evaluated with the neighborhood information in CUDB.

Note that the intersection (throat) is simply a part of the two projections involved, as shown in Figure 4.2(b). Both adjacent boxes may be large enough to contain the current ball; however, the ball cannot pass from one box to the other because the throat between them is smaller than the current diameter. The existing throat when $\overline{\beta_i}^A \subseteq \overline{\beta_j}^A$ or $\overline{\beta_j}^A \subseteq \overline{\beta_i}^A$ (see Figure 4.2(a)) can be ignored because both boxes are large enough to contain the intruded fluid as they have survived the discarding process.

Algorithm 8 shows the steps of the orthogonal throat detection process for a single ABC-sorting.

Algorithm 8: detectOrthogonalThroats

```
Input  : $Q$;                          /* ABC-sorted CUDB-represented pore space /*
Input  : $D$;                                                      /* Diameter /*
Output: $ortogonals$;                                /* Set of orthogonal throats /*
$ortogonals \leftarrow \emptyset$;
for all $\beta_i \in Q$ do
    for all $\beta_j \in \beta_i.AFN$ do                  /* All A-forward neighbors /*
        $throat \leftarrow \overline{\beta_i}^A \cap \overline{\beta_j}^A$;
        if $length_C(throat) < D$ then
            Add $throat$ to $ortogonals$;
        end if
    end for
end for
```

[1] A set is called an open set if it does not contain any of its boundary points.

Orthogonal Throats Creation

An orthogonal throat is represented by a rectangle perpendicular to the A-coordinate. Thus, an orthogonal throat is defined as a structure containing five coordinate values:

$$orthogonalThroat = \{a, b_0, c_0, b_1, c_1\}$$

where a represents the throat position in the A-coordinate, and b_0, b_1, c_1 and c_0 the coordinates of its two diagonally opposed vertices in the B and C-direction respectively. Then, for each orthogonal throat, an EVM cuboid of dimensions $1 \times |b_1 - b_0| \times |c_1 - c_0|$ is created.

2D Oblique Throats Detection

For clarity purposes, the 2D case is analyzed first. Notice that an oblique 2D throat is an oblique segment. The next process generates the oblique throats for an AB-sorted CUDB, which are the same generated for the corresponding BA-sorted CUDB.

Let β_{i-1}, β_i and β_{i+1} be three consecutive A-adjacent boxes in an AB-sorted 2D CUDB, and let $\overline{\beta_i}^A$ be the open set of the β_i projection over a segment perpendicular to the A-coordinate. Then, an oblique throat exists between β_{i-1} and β_{i+1} if the following conditions are satisfied:

$$(\overline{\beta_{i-1}}^A \cap \overline{\beta_{i+1}}^A) \neq \emptyset \tag{4.2a}$$

$$\overline{\beta_{i-1}}^A \not\subseteq \overline{\beta_{i+1}}^A \tag{4.2b}$$

$$\overline{\beta_{i-1}}^A \not\supseteq \overline{\beta_{i+1}}^A \tag{4.2c}$$

$$(\overline{\beta_{i-1}}^A \cap \overline{\beta_{i+1}}^A) \subset \overline{\beta_i}^A \tag{4.2d}$$

$$diagonal(\beta_{i-1}, \beta_{i+1}) < D \tag{4.2e}$$

where D is the diameter corresponding to the current intruded pressure and $diagonal()$ returns the length of the diagonal of the rectangle defined by $\overline{\beta_{i-1}}^A \cap \overline{\beta_{i+1}}^A$ and the distance between both boxes in the A-direction. Figure 4.3(a) illustrates the above conditions.

However, oblique throats are not restricted to three consecutive boxes. Depending on the geometry of the object, an oblique throat can occur between two boxes β_S and β_T which have

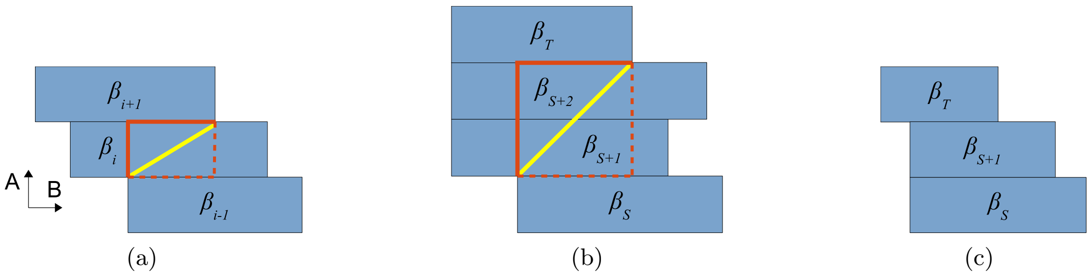


Figure 4.3: Oblique throats. (a) Simplest case of oblique throat (with just one intermediate box). (b) Oblique throat with more than one intermediate box. (c) Non oblique throat.

more than one intermediate box between them, as shows Figure 4.3(b). In these cases, all of the conditions in Expression 4.2 must be satisfied for β_S and β_T instead of β_{i-1} and β_{i+1} respectively. Therefore, these conditions are rewritten as:

$$(\overline{\beta_S}^A \cap \overline{\beta_T}^A) \neq \emptyset \tag{4.3a}$$

$$\overline{\beta_S}^A \not\subseteq \overline{\beta_T}^A \tag{4.3b}$$

$$\overline{\beta_S}^A \not\supseteq \overline{\beta_T}^A \tag{4.3c}$$

$$(\overline{\beta_S}^A \cap \overline{\beta_T}^A) \subset \overline{\beta_i}^A,\ \forall i \mid S < i < T \tag{4.3d}$$

$$diagonal(\beta_S, \beta_T) < D \tag{4.3e}$$

Condition 4.3d means that $\overline{\beta_S}^A \cap \overline{\beta_T}^A$ must be contained in every intermediate box projection between β_S and β_T. In addition, Figure 4.3(c) shows why it is necessary to consider open sets for conditions of Expression 4.3. Otherwise, the aforementioned conditions would be satisfied, and the algorithm would detect oblique throats where none exist.

The previous expressions are used to detect a 2D oblique throat between β_S and β_T. Although a rectangle is actually obtained (see the rectangle highlighted in orange color in Figures 4.3(a) and 4.3(b)), there are two options to create the oblique throat. The first one is totally consistent with the orthogonal feature of the CUDB-based approach, and the throat is computed as a L-shaped object. There are two possible L-shaped objects showed in continuous and dashed orange lines respectively in Figures 4.3(a) and 4.3(b), and we can choose any of them. The second strategy to compute the throat is by considering the diagonal included in the mentioned rectangle (see the inclined line highlighted in yellow color in Figures 4.3(a) and 4.3(b)). Both strategies separate the two regions in the current scan direction, but, as the diagonal fairly cuts the narrowing zone between the two involved regions, this strategy is used in the final version of the presented method.

A box β_S can generate several oblique throats with its consecutive boxes in the A-direction (see Figure 4.4(a)). Therefore, in order to detect all the possible oblique throats, all consecutive boxes of β_S must be scanned. This process searches for any configuration that satisfies conditions of Expression 4.3.

Conditions 4.3a and 4.3c are required conditions to stop the search process, while the others are not. The search stops in case that Condition 4.3a is not satisfied, as every subsequent box β_T will not satisfy Condition 4.3d in the following iterations (see Figure 4.4(b)). When Condition 4.3a is not satisfied, every subsequent box β_T will not satisfy either Condition 4.3b or 4.3d in the following iterations (see Figure 4.4(c)). Moreover, the search must be also stopped when the distance between $\beta_S.V1$ and $\beta_T.V1$ in the A-coordinate is greater than D, as the subsequent oblique throats will be also greater than D. However, when conditions 4.3b and 4.3d are not satisfied, there may still be subsequent boxes that define throats that satisfy all conditions of Expression 4.3, see Figures 4.4(d) and 4.4(e).

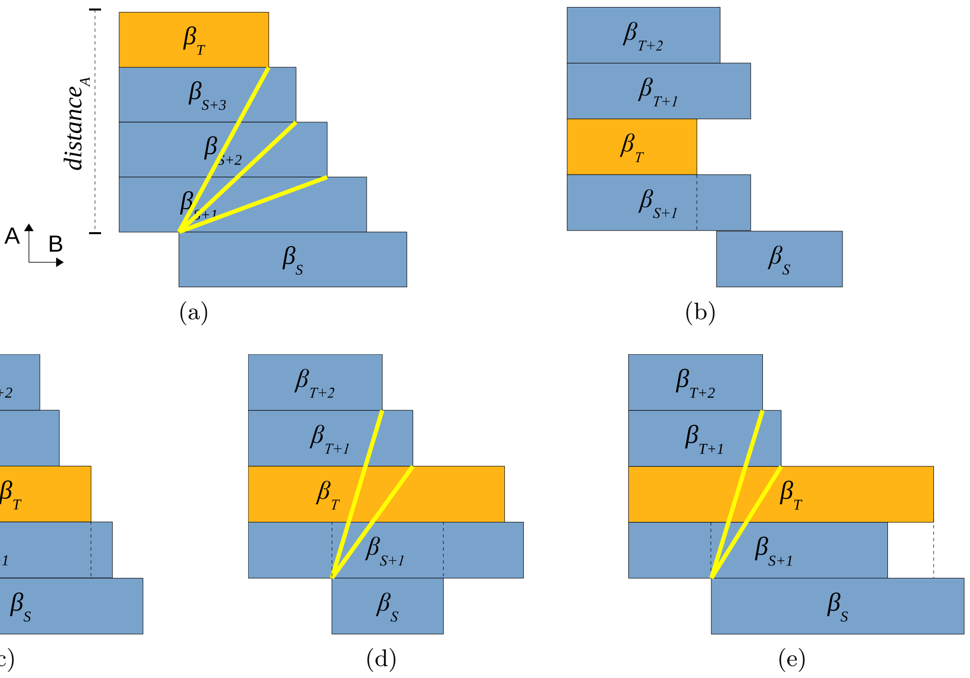


Figure 4.4: 2D oblique throats scanning process. (a) A box that generate several oblique throats. (b) Expression 4.3a is not satisfied. (c) Expression 4.3c is not satisfied. (d) Expression 4.3b is not satisfied. (e) Expression 4.3d is not satisfied.

3D Oblique Throats Detection

For a 3D ABC-sorted CUDB, the same conditions of Expression 4.3 must be satisfied, but in this case, box projections are over a plane perpendicular to the B-coordinate. This process generates the oblique throats for an ABC-sorted CUDB, which are the same generated for the corresponding ACB-sorted CUDB. Therefore, three ABC-sorted CUDB encodings of the pore space with different A-coordinate are required to detect all oblique throats.

In 3D, the oblique throat formed between β_S and β_T is actually an inclined rectangle (see Figure 4.2(c)). This rectangle has a pair of inclined edges and a pair of orthogonal ones, where the orthogonals ones are A-axis aligned. The length of the inclined edges is considered the length of the throat, so, function *diagonal*() returns this value. The length of the orthogonal edges is not considered because the detected throat could be part of a longest throat in the A-direction; however, this length must be adjusted according to the A-edge length of every intermediate box, since the throat must be inside all of them.

Figure 4.5(a) shows a set of boxes with ZYX-sorting and the first step of the oblique throat detection. In this case, the inclined edges are computed based on the Y-axis. Figure 4.5(b) shows another view of the boxes and the resulting oblique throat. Note in this case that, the length of the orthogonal edges is determined by the length in the Z-direction of every intermediate box. Both images show that all of the conditions in Expression 4.3 are satisfied.

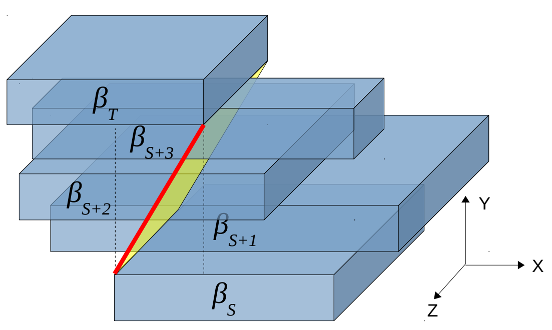
Algorithms 9 and 10 show the steps of the oblique throat detection using a recursive strategy for a single ABC-sorting.

Algorithm 9: detectObliqueThroats

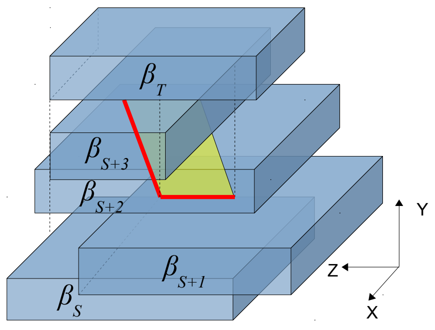
```
Input  : $Q$;                    /* ABC-sorted CUDB-represented pore space /*
Input  : $D$;                                                /* Diameter /*
Output: $obliques$;                           /* Set of oblique throats /*
$obliques \leftarrow \emptyset$;
$I \leftarrow \emptyset$;                  /* Stack of intermediate boxes pointers /*
for all $\beta_S \in Q$ do
    for all $\beta_i \in \beta_S.BFN$ do                /* All B-forward neighbors /*
        if $\overline{\beta_S}^B \not\supseteq \overline{\beta_i}^B$ then       /* Condition 4.3c and start recursion /*
            $I$.push($\beta_i$);
            $detectObliques(\beta_S, I, D, obliques)$;
            $I$.pop();
        end if
    end for
end for
```

Algorithm 10: detectObliques

```
Input  : $\beta_S$;                                          /* The base box /*
Input  : $I$;                                /* Stack of intermediate boxes /*
Input  : $D$;                                                /* Diameter /*
Input  : $obliques$;                          /* Set of oblique throats /*
$\beta_{top} \leftarrow I$.top();
if $distance_B(\beta_S, \beta_{top}) > D$ then
    return;                                               /* Stop process /*
end if
for all $\beta_T \in \beta_{top}.BFN$ do                /* All B-forward neighbors /*
    if $(\overline{\beta_S}^B \cap \overline{\beta_T}^B) \neq \emptyset$ and $\overline{\beta_S}^B \not\supseteq \overline{\beta_T}^B$ then      /* Conditions 4.3a and 4.3c /*
        $validThroat \leftarrow$true; $throat \leftarrow$ Create oblique throat from $\beta_S$ to $\beta_T$;
        if $\overline{\beta_S}^B \subseteq \overline{\beta_T}^B$ and $diagonal(\beta_S, \beta_T) \geq D$ then    /* Cond. 4.3b and 4.3e /*
            $validThroat \leftarrow$false;
        end if
        if $validThroat$ then
            for all $\beta_i \in I$ do
                if $(\overline{\beta_S}^B \cap \overline{\beta_T}^B) \not\subset \overline{\beta_i}^B$ then               /* Condition 4.3d /*
                    $validThroat \leftarrow$false; break;
                end if
                Adjust $throat$ length in the A-direction according to $\overline{\beta_i}^B$ length;
            end for
            if $validThroat$ then Add $throat$ to $obliques$; end if;
        end if
        $I$.push($\beta_T$);
        detectObliques($\beta_S, I, D, obliques$);
    end if
end for
```



(a)



(b)

Figure 4.5: ZYX-sorted set of boxes. (a) First step of oblique throat detection. (b) The resultant oblique throat (rotated -90°around Y).

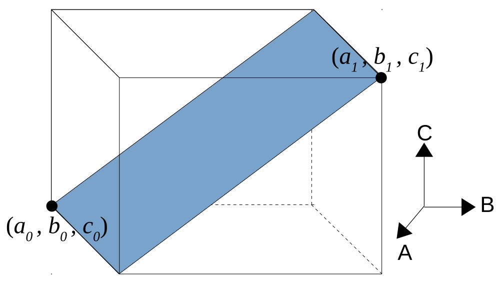
Oblique Throats Creation

In order to save the necessary information to create an EVM-represented oblique throat, it is defined as a structure containing six coordinate values. These coordinates represent the parallelepiped containing the oblique throat (see Figure 4.6(a)):

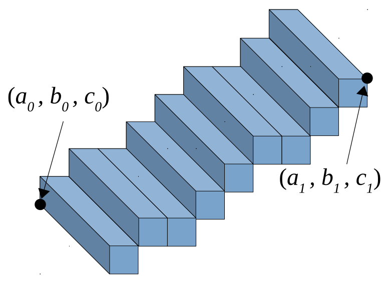
$$obliqueThroat = \{a_0, b_0, c_0, a_1, b_1, c_1\}$$

where a_0 and a_1 define the length of the orthogonal edges, and the remaining ones define the length of the inclined edges. In order to know the oblique throat orientation inside the parallelepiped, we ensure that $b_0 < b_1$.

To create an EVM-represented oblique throat, the Bresenham's algorithm [20] has been used to determine the points that must be outlined in a square grid to form a close approximation to the straight line representing the throat. This implementation receives the coordinates b_0, c_0, b_1, and c_1 and generates a set of points. Then, for each point a cuboid with dimensions $|a_1 - a_0| \times 1 \times 1$ is created. Therefore, an oblique throat is the result of the union of all the created cuboids. As these cuboids are disjoint each other, according to the EVM Property 5 (see Section 2.2.4), an XOR operation is performed. Figure 4.6(b) shows an example of an oblique throat creation.



(a)



(b)

Figure 4.6: Oblique throat creation example.

4.2.4 Narrow Throats Removal

Each computed throat separates the remaining pore space into two disjoint pieces, so removing a throat stops the mercury passing from an invaded region to an adjacent one through the too small removed throat. Therefore, once all EVM-represented orthogonal and oblique throats have been created, the union of all of them in a single EVM-represented object must be performed. Then, the difference between the EVM-represented pore space and the EVM-represented throats produces the object partitioned along the corresponding throats.

Figure 4.7 illustrates a configuration that produces three overlapping oblique throats, shown in Figure 4.2(d). It also shows the union of these throats and the resulting partitioning of the object as the difference between it and the union of throats. As the oblique throat is a subset of the pore space, in order to perform the difference Boolean operation, the EVM Property 6 is used and an XOR operation is performed instead.

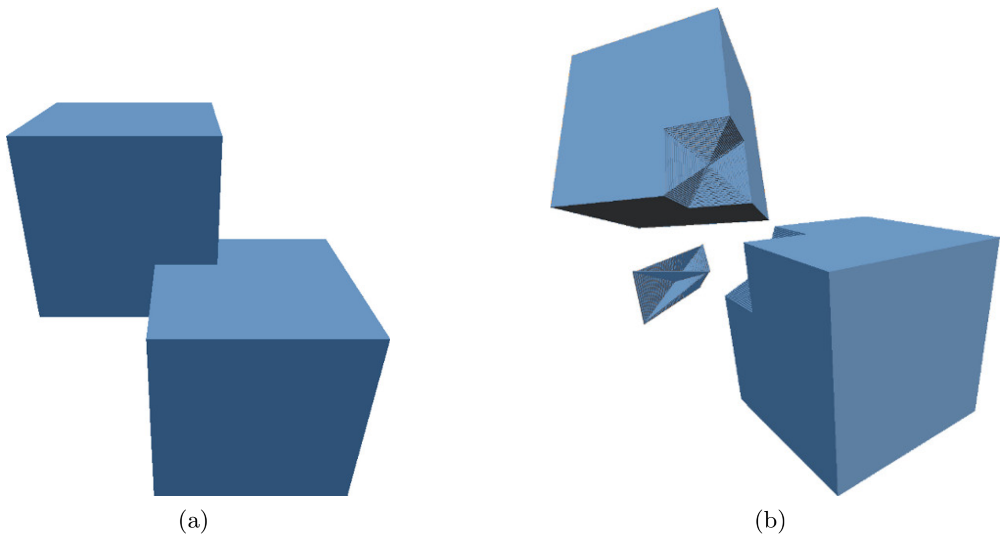


Figure 4.7: (a) Configuration that produces the intersection of three oblique throats (shown at Figure 4.2(d)). (b): Removed part and disjoint components obtained (the two blocks has been separated and rotated for more clearness).

4.2.5 Mercury Intrusion Simulation

After the discarding and throats removal processes, the remaining pore space partitioned in multiple disconnected regions is ready to simulate mercury intrusion. In the presented method, the set of entry points is defined as those boxes labeled with a predefined label value $ENTRY$. This step can be performed in a preprocessing, but, for a faster simulation, those boxes which touch the boundary of the AABB of the solid space can be considered as entry points.

Algorithm 11 shows the steps of the simulation process for a given pore space P and diameter D. Function $simulateIntrusion()$ receives as input an EVM-represented pore space, which is used to create the three CUDB encodings required by the discarding and throat detection processes, and returns an EVM object E that represents the pore space after the discarding and throats removal processes. Internal conversions from CUDB to EVM are necessary to sort the vertices and get the corresponding CUDB representations, and to perform the Boolean operations that remove the throats.

Finally, Algorithm 12 describes the iterative process for the whole CUDB-based MIP simulation. The main function, *porosimeter*(), evaluates all diameters D_i, $1 \leq i \leq n$ in order to simulate the corresponding mercury intrusion. After each intrusion simulation, function *detectEntryPoints*() labels the corresponding boxes as $ENTRY$, then, a CCL is applied to

Algorithm 11: simulateIntrusion

```
Input  : $P$;                                    /* EVM-represented pore space /*
Input  : $D$;                                                   /* Diameter /*
Output: $E$;                        /* EVM-represented partitioned pore space /*
/* ---Discarding process---                                                /*
$E \leftarrow P$;
for each sort in (XYZ,YZX,ZXY) do
    $E.setSorting(sort)$;  $Q \leftarrow EVMtoCUDB(E)$;
    Remove from $Q$ boxes with C-edge length $< D$;
    $E \leftarrow CUDBtoEVM(Q)$;
end for
/* ---Throats detection process---                                         /*
$ortThroats \leftarrow \emptyset$;  $oblThroats \leftarrow \emptyset$;  /* Sets of orthogonal and oblique throats /*
for each sort in (XYZ,YZX,ZXY) do
    $E.setSorting(sort)$;  $Q \leftarrow EVMtoCUDB(E)$;
    $ort \leftarrow detectOrtogonalThroats(Q, D)$;  Add $ort$ to $ortThroats$;
    $obl \leftarrow detectObliqueThroats(Q, D)$;  Add $obl$ to $oblThroats$;
end for
/* ---Throats removal process---                                           /*
$ortEVM \leftarrow createEVMOrtThroats(ortThroats)$;
$oblEVM \leftarrow createEVMOblThroats(oblThroats)$;
$E \leftarrow E \otimes^* (ortEVM \cup^* oblEVM)$;
```

Algorithm 12: porosimeter

```
Input  : $P$;                                    /* EVM represented pore space /*
Output: $\mathcal{R}$;                      /* Set of CUDB-represented pore regions /*
$current \leftarrow P$;
for $D = D_1$ to $D_n$ do
    $E \leftarrow simulateMIP(current, D)$;
    /* ---Intrusion simulation process---                                  /*
    $invadedCUDB \leftarrow EVMtoCUDB(E)$;
    $detectEntryPoints()$;
    CCL($invadedCUDB$);
    Remove those CC of $invadedCUDB$ no corresponding to entry regions;
    Add $invadedCUDB$ to $\mathcal{R}$;
    $current \leftarrow E$;
end for
```

the partitioned pore space. Any connected component (CC) having a box labeled as $ENTRY$ represents an entry CC. Then, those CC which are not entry components are removed because they represent inaccessible regions. Thus, the remaining boxes represent the region invaded by the diameter D. This invaded region is stored in a set of CUDB-represented pore regions $\hat{R}$, in order to compute the pore map. Note that after each iteration, the partitioned pore space, after the corresponding discarding and throats removal processes, is used in the next iteration, as those regions inaccessible by the current diameter are also inaccessible by larger diameters.

4.2.6 Pore Map Computation

Once the iterative process has finished, a set of n labeled CUDB-represented regions $\mathcal{R} = \{R_1, R_2, \dots, R_n\}$ have been produced. Each region R_i corresponds to the subset of the pore space invaded by the mercury at the intrusion pressure ρ_i, so, according to the Washburn law, the following property is always satisfied:

$$\rho_i < \rho_j \Rightarrow D_i > D_j \Rightarrow R_i \subseteq R_j, \ \forall i < j \tag{4.4}$$

Then, let $\hat{R}_i$ be the region invaded by the mercury exclusively at the intrusion pressure ρ_i , i.e., the region that is not invaded at any other lower pressure. For every ρ_i, $\hat{R}_i$ can be obtained in terms of the CUDB-represented regions as:

$$\hat{R}_1 = R_1 \tag{4.5a}$$

$$\hat{R}_i = R_i - R_{i-1}, \quad \forall i = 2, \dots, n \tag{4.5b}$$

Finally, the pore map, PM, in which the entire pore space is shown subdivided into pore regions, each represented by a set of boxes, is given by:

$$PM = \bigcup_{i=1}^{n} \hat{R}_i \tag{4.6}$$

Figure 4.8 illustrates a detail of the resulting pore map for a 2D sample. Let us consider

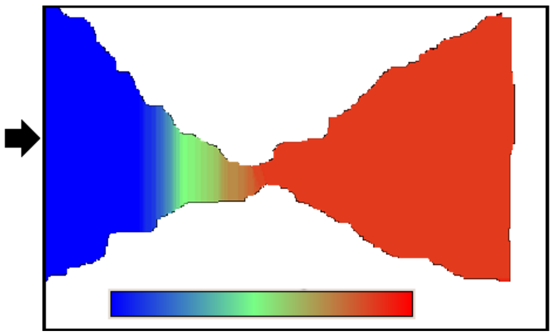


Figure 4.8: Detail of a porosimetry.

that mercury enters from the left side of the image. Several diameters can be identified in the figure. Moreover, it can be seen that the narrow throat at the middle of the sample prevents the mercury from reaching the right side of the pore space at lower input pressures (in blue).

4.2.7 Virtual Porosimeter Results

The CUDB-based method, like those based on prior computation of the skeleton, is a geometric approach that computes the expected solution. Figure 4.9 shows two phantom datasets for which it is easy to determine the theoretical solution. The example in (a) has only one entry point at the top of the object, while in (b) there are two entry points. Because MIP actually computes the volume associated with the narrowings of the object, the expected results are different as shown in these figures. The example in (c) shows a phantom scaffold with its expected theoretical result, where the entry points are all those touching the boundary.

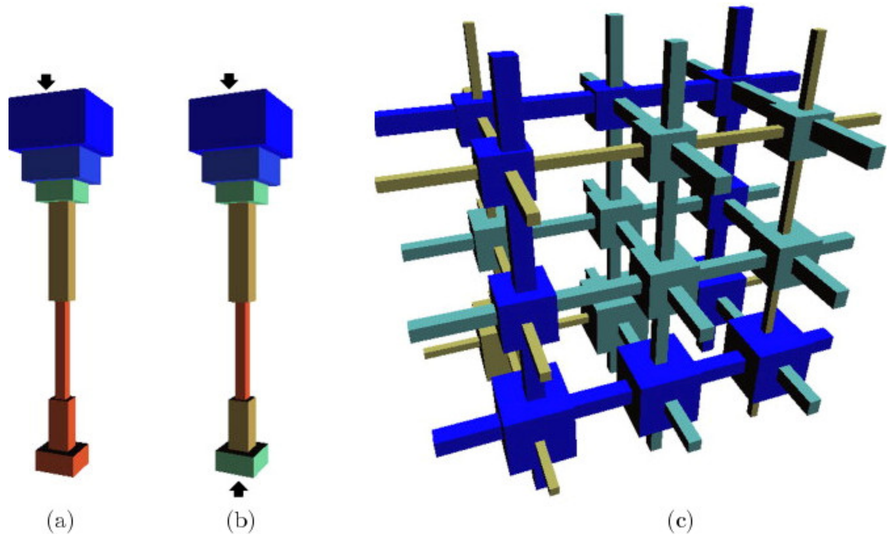


Figure 4.9: Pore map of phantom datasets. (a) and (b) A single object. (c) A scaffold.

The CUDB-based approach has been compared with an skeleton-based approach where the skeleton voxels are labeled with the chessboard distance [170]. For the two phantom datasets in Figure 4.9 the result is the same for both and matches the expected theoretical result.

For real datasets, the skeleton-based approach can have up to a one-voxel error, due to the medialness property of the skeleton for even distances in a discrete space. As the CUDB-based approach requires no skeleton, it avoids such approximation errors.

Figure 4.10 shows a visual comparison of a 2D section of a real sample corresponding to a porous biomaterial sample (PLA) with multiple entry points. Figure 4.10(c) shows the resulting pore map obtained with the skeleton-based virtual MIP, while the other two images show the result with the CUDB-based method. Figure 4.10(a) shows the result using the inclined rectangle strategy for oblique throats creation, and Figure 4.10(b) the result using the L-shape strategy (for similarity to skeleton-based). In this figure, a slight color difference between (a) and (c) is noted, while (b) shows almost the same colors than (c).

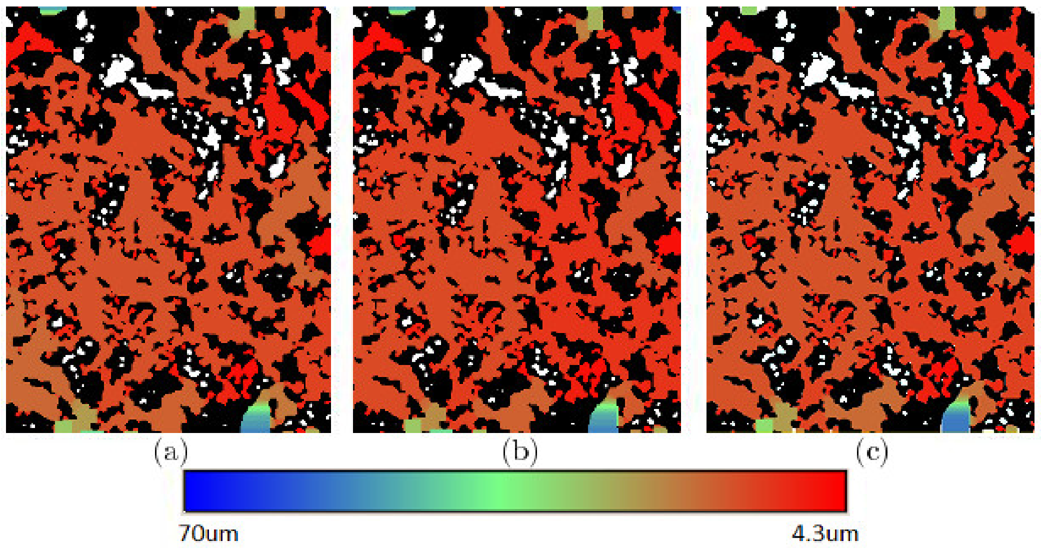


Figure 4.10: Pore map resulting after MIP simulation. (a) and (b) with the CUDB-based method using the inclined rectangle and the L-shape strategy respectively, and (c) With the skeleton-based method.

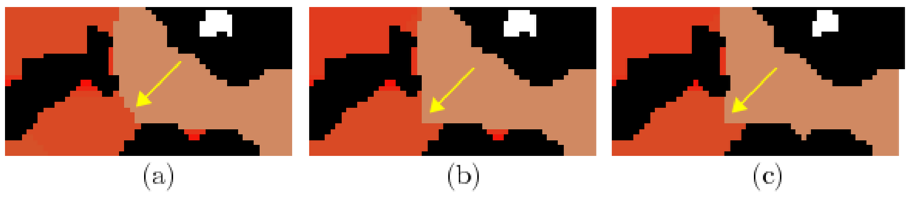
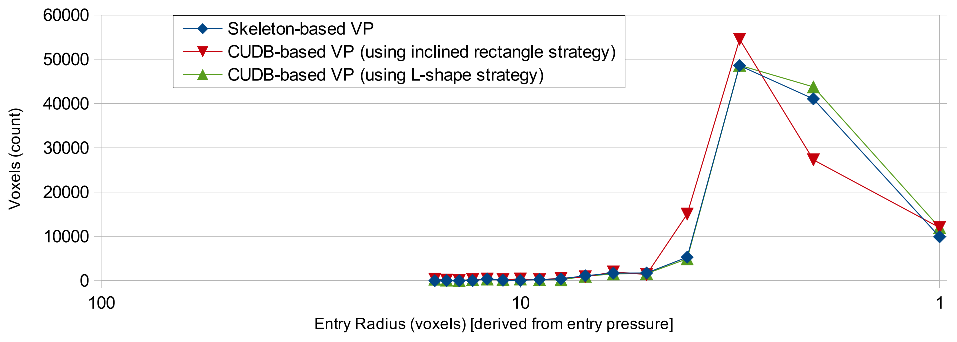


Figure 4.11: Oblique throat shape. (a) and (b) with the CUDB-based method using the inclined rectangle and the L-shape strategy respectively, and (c) with the skeleton-based method.

For a more objective test, the pore-size distribution of the above 2D sample was computed using the same three approaches (see Figure 4.12). As expected, the L-shape strategy gives almost the same result than the skeleton-based approach. In this case, the difference is only due to the medialness approximation used in the skeleton-based approach. On the other hand, comparing these results with the result that uses the inclined rectangle strategy gives more differences, due to the different volume associated to the two regions separated by an oblique throat. Figure 4.11 shows a zoom of the same region of the three cases of Figure 4.10, where the shape of an oblique throat is depicted in detail.

From the analysis of the absolute-volume histogram in Figure 4.12 with the inclined rectangle strategy for oblique throats, the maximum difference computed between the CUDB-based and skeleton-based approach is less than 12% with respect to the whole intruded volume. However if the L-shape strategy is used, this maximum difference is less than 3%. These results are consistent with the fact that the L-shape strategy is geometrically more similar to the skeleton-based method than the inclined rectangle strategy.



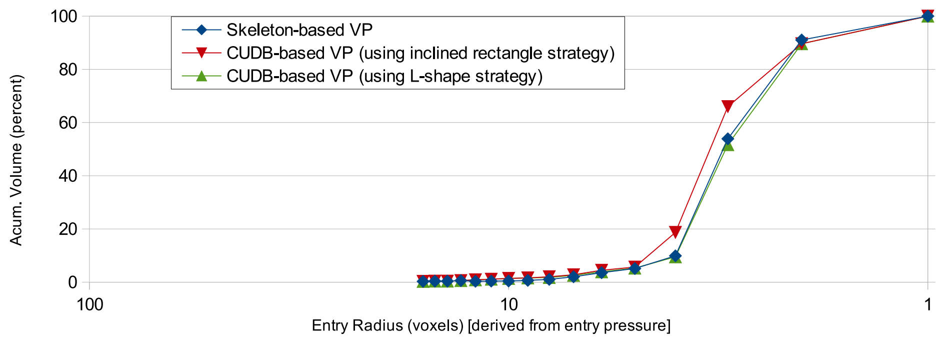
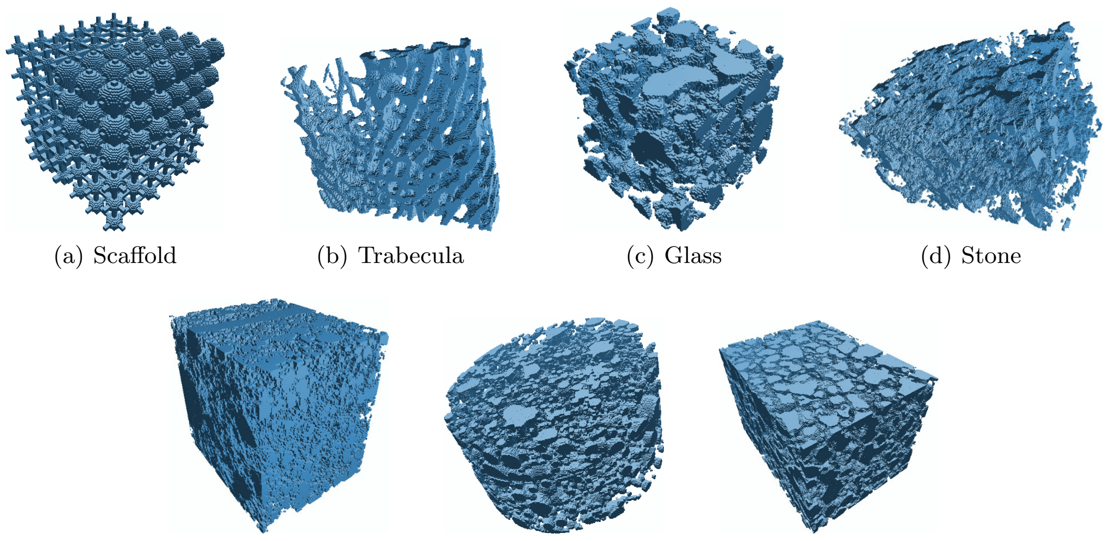


Figure 4.12: Absolute (top) and relative (bottom) pore-size distribution histograms obtained with the CUDB-based and skeleton-based MIP methods.

The main goal of the presented CUDB-based method is its performance. Therefore, running times with the skeleton-based method [170] have been compared in a collection of both phantom and real 3D porous samples. Rendered images of the corresponding pore spaces are shown in Figure 4.13 and described below.

- Scaffold. A synthetic scaffold.
- Trabecula. A synthetic trabecula fragment.
- Glass: A glass sample consisting of calcium phosphate glass with a 7.4 μm^3 voxel resolution.
- Stone: A stone sample consisting of sedimentary rock from a Lybian oil-bearing unit with a 4.4 μm^3 voxel resolution.
- Soygel: Sample of hydroxyapatite with a soy-based foaming agent for bone prosthetics with a 16 μm^3 voxel resolution.
- Tween: Sample of hydroxyapatite with Tween foaming agent for bone prosthetics with a 16 μm^3 voxel resolution.
- PLA: A biomaterial sample consisting of polylactic acid (PLA) with a 16 μm^3 voxel resolution.


(a) Scaffold (b) Trabecula (c) Glass (d) Stone

(e) Soygel (f) Tween (g) PLA

Figure 4.13: Rendered images of the pore space of the test datasets.

All the real samples have been scanned by `Trabeculae`® and segmented by thresholding and applying noise filtering. To illustrate the relative speed of the CUDB-based MIP method, Table 4.1 shows the run times for both approaches, skeleton-based and CUDB-based, using the aforementioned samples. The corresponding programs have been written in C++ and tested on a PC Intel®Core 2 Duo CPU E6600@2.40GHz with 3.2 GB RAM with Linux.

Figure 4.14 shows a simulation of the mercury intrusion at three different input pressures and the resulting 3D pore map for the Stone and Tween sample.

In regard to the comparison of the results with those of real MIP, we had been provided with the physical porosimetry results of the biomaterial sample, PLA, and Figure 4.15 shows the corresponding histogram (top) together with the histogram obtained with the CUDB-based method (bottom). The shaded area in the top histogram indicates the diameter range that cannot be reached by the CUDB-based method due to the resolution of the μCT device.

Table 4.1: MIP run time comparison. For each dataset: size of the voxel model and, run time (in seconds) for skeleton-based (Skeleton computation and intrusion) and CUDB-based MIP approaches.

Dataset	Size	Skeleton-based			CUDB-based
		Sk. Comp.	MIP	Total	
Scaffold	130x130x126	61.8	67.2	129.0	2.0
Trabecula	138x163x37	211.9	13.4	225.3	11.8
Glass	100x100x100	55.3	7.8	63.1	3.9
Stone	159x271x179	221.2	132.0	353.2	39.7
Soygel	251x251x206	1918.0	78.0	1996.0	446.2
Tween	376x375x206	2527.0	114.0	2641.0	230.6
PLA	237x220x314	2191.6	607.0	2798.6	271.7

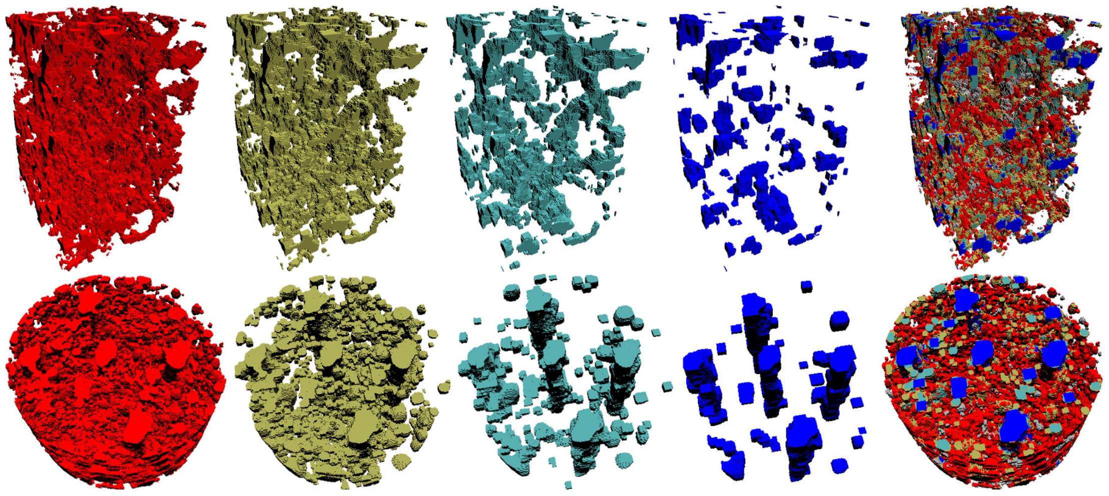
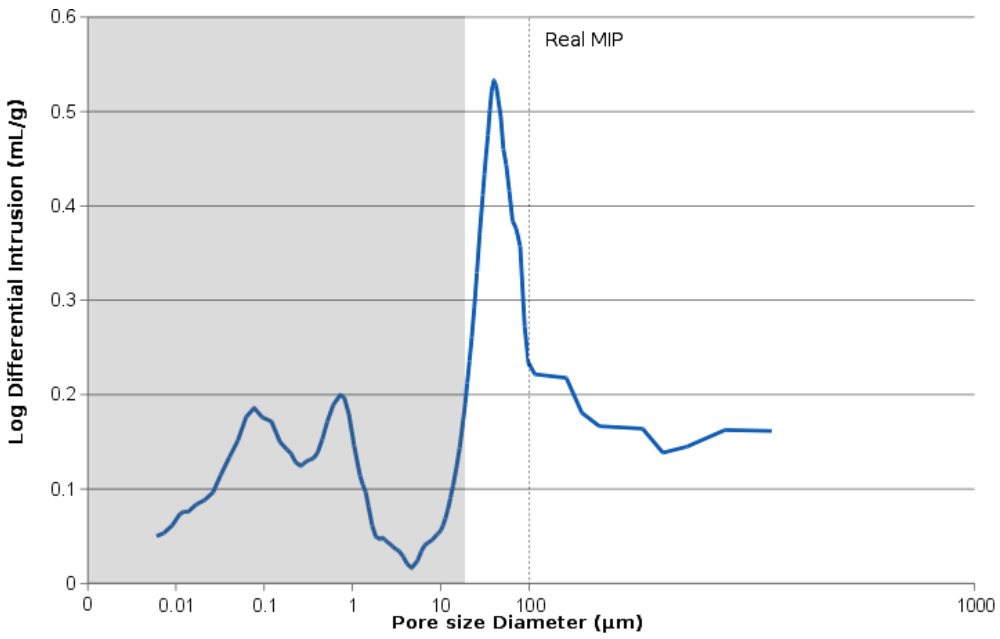


Figure 4.14: Simulation of progressive mercury intrusion over the Stone (top) and Tween (bottom) samples. From left to right: original pore space; invaded region at higher pressure; invaded region at medium pressure; invaded region at lower pressure; putting all together.

Based on the analysis of these histograms with the aid of an expert, the conclusion is that the diameter value corresponding to the maximum volume (which is relevant information for materials scientists) is almost the same. Moreover, despite the different natures of the experiments, which makes a total direct correlation difficult, the behavior of the histograms is quite similar.

A more detailed discussion of the comparison of physical and *in-silico* porosimetry methods is beyond the scope of this thesis. However, a related and extensive discussion applied to sedimentary rock samples [171], as well as a discussion of the problems and challenges of physical and *in-silico* methods applied to the study of the pore-size distribution of porous materials (such as concrete) [26] have been presented.



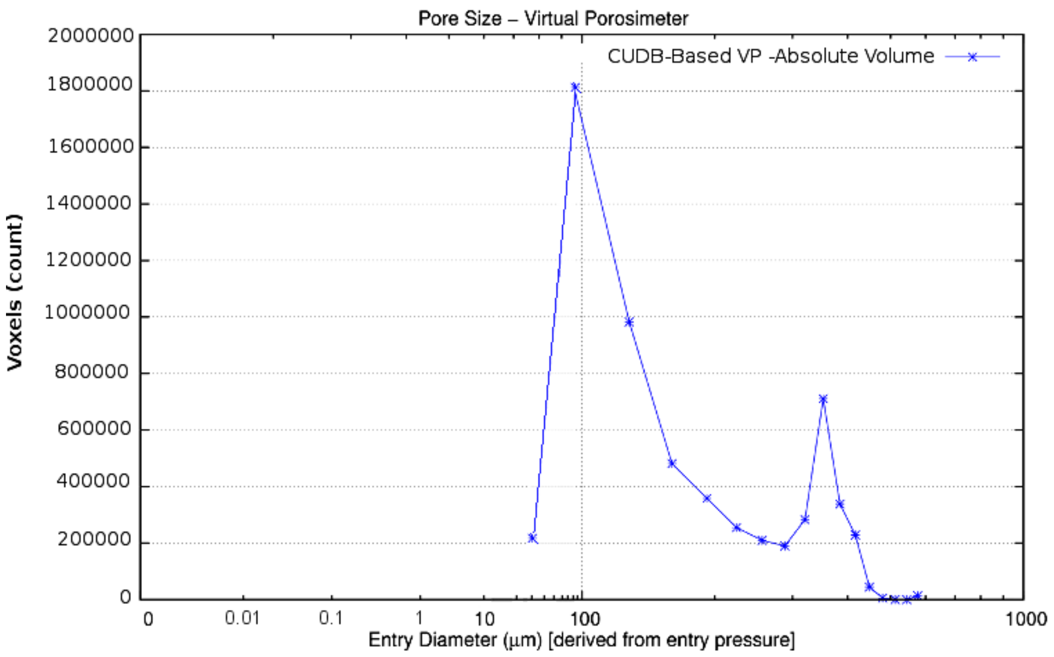


Figure 4.15: Pore-size distribution histograms for the PLA sample obtained with real (top) and the CUDB-based virtual (bottom) MIP methods.

4.3 Connectivity

In this section, a method to compute the Euler characteristic, χ, and the genus of binary volume datasets represented with the CUDB model is presented. The method is derived from the classical method used with a voxel model (see Section 2.3.3). The computation of χ and the genus is achieved by counting the number of unitary basic elements – vertices, edges, faces and voxels – with which a box of the CUDB model contributes, taking into account the overlapping regions among them and using the CCL process.

4.3.1 Method Overview

The method presented in this thesis applies Expression 4.7 to each box of the CUDB to compute the number of voxels (n_3), face (n_2), edges (n_1) and vertices (n_0), taking into account the overlapping regions among boxes. Then, with the CUDB-CCL process, the connected components (h_0) and cavities (h_2) of the object are obtained in order to compute the connectivity (h_1), which is related to the genus, from Expression 4.8.

$$\chi = n_0 - n_1 + n_2 - n_3 \tag{4.7}$$

$$\chi = h_0 - h_1 + h_2 \tag{4.8}$$

Although binary volumes with adjacency pairs (6, 26) or (26, 6) are non-manifold, χ and the genus can be computed unambiguously for them. When computing n_0 and n_1 in Expression 4.7, the adjacency pair must be taken into consideration in such a way that non-manifold edges and vertices are counted once for 26-adjacency and twice for 6-adjacency. For example, in the case of the object depicted in Figure 4.16, considering the adjacency pair (26, 6), the number of connected components (h_0) is 1, and vertices v_1 to v_6 and edges e_1, e_2 must be counted just once because they belong to two connected voxels. Then, in this case, the genus (h_1), which can be seen as the number of handles, is 2. On the other hand, considering the adjacency pair (6, 26), $h_0 = 3$, and vertices v_1 to v_6 and edges e_1, e_2 must be counted twice because they belong to two disjoint voxels. In this case, the genus is 0.

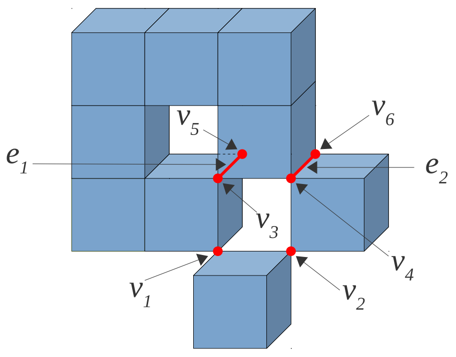


Figure 4.16: Illustrative figure consisting of 9 voxels to compute the genus depending on the selected adjacency pair.

In the voxel model, a simple way to compute the number of faces, edges and vertices reported for each voxel is by checking the lower 13 neighbors (N^-) of the voxel for a backward scan (see Figure 4.17), where the 6 faces, 12 edges and 8 vertices of each visited voxel are added, and the possible shared elements (3 faces, 9 edges and 7 vertices) are subtracted. An analogy to this reasoning is used in the method presented here.

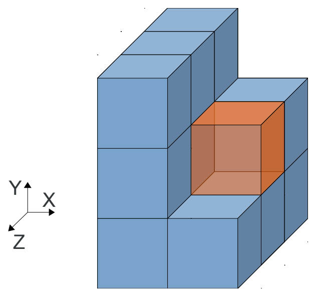


Figure 4.17: The lower 13 neighbors (N^-) of a voxel.

It is important to point out that, independently of the used adjacency pair, (26, 6) or (6, 26), in the binary volume model, the proposed method requires a CUDB model with the neighboring boxes information according to 26-adjacency. This is because when a 6-adjacency is considered, the connected components may have boxes that are 26-adjacent. So, in order to compute χ and the genus for the (6, 26) adjacency pair, a preprocesing is performed to get the connected component with 6-adjacency. Then, each connected component is separately analyzed with 26-adjacency to count the enclosed elements. In the case of the (26, 6) adjacency pair, the method is applied directly. So, henceforth, let us focus on the case of the (26, 6) adjacency pair. The proposed method can be applied with any ABC-ordering of the CUDB model.

4.3.2 Connectivity Computation

The Euler characteristic and the genus is computed based on Expression 4.7 considering a CUDB box as a rectangular prism enclosing a finite number of voxels. Let β be a box in the CUDB model, and let d_x, d_y and d_z its side lengths. Then, the number of enclosed voxels (γ_β), faces (f_β), edges (e_β) and vertices (v_β) of β are computed as:

$$\gamma_\beta = d_x d_y d_z \tag{4.9}$$

$$f_\beta = [d_x d_y (d_z + 1)] + [d_x (d_y + 1) d_z] + [(d_x + 1) d_y d_z] \tag{4.10}$$

$$e_\beta = [d_x (d_y + 1)(d_z + 1)] + [(d_x + 1) d_y (d_z + 1)] \tag{4.11}$$

$$+ [(d_x + 1)(d_y + 1) d_z] \tag{4.12}$$

$$v_\beta = (d_x + 1)(d_y + 1)(d_z + 1) \tag{4.13}$$

A traversal of CUDB is performed and for each box β, its unitary elements (γ_β, f_β, e_β, v_β) are computed according to Expressions 4.9 to 4.13. However, there are overlapping regions among boxes and the method must deal correctly with them. To do so, for each box β, its backward neighbors (ABN and BBN) must be analyzed in order to subtract the elements reported by the overlapping regions.

When an ABC-sorted CUDB model is generated considering the 26-adjacency, boxes in a non-manifold configuration are neighbors only in one direction. Two edge-adjacent boxes β_i and β_k are neighbors in the A-direction if the overlapping region between them is a segment (part of an edge) B or C-aligned (see Figure 4.18(a) and (b)), but if the segment is A-aligned, β_i and β_k are neighbors in the B-direction (see Figure 4.18(c)). Two vertex-adjacent boxes β_i and β_k, are neighbors only in the A-direction (see Figure 4.18(d)).

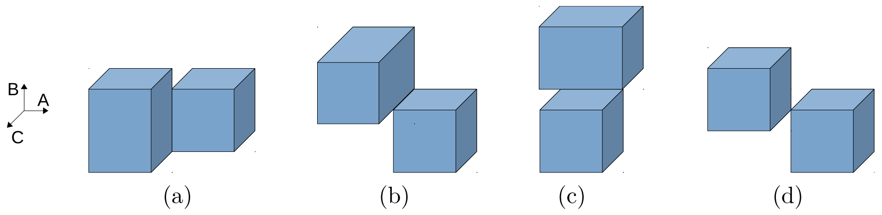


Figure 4.18: Boxes in a non-manifold configuration.

Shared Elements Computation

Overlapping regions can be rectangles (2D), line segments (1D) (segments from now on) or points (0D). They have to be detected and their contribution computed and added or subtracted to the global value.

Every box β_i shares a rectangle R with any box $\beta_k \in (\beta_i.ABN \cup \beta_i.BBN)$ (see Figure 4.19(a) and (b) in red). The basic unitary elements enclosed by this rectangle are computed twice and therefore they must be subtracted once. Let r_x and r_y be the side lengths of R, the corresponding number of faces (f_R), edges (e_R) and vertices (v_R) can be be computed in a way similar to that of Expressions 4.10 to 4.13:

$$f_R = r_x r_y \tag{4.14}$$

$$e_R = r_x(r_y + 1) + (r_x + 1)r_y \tag{4.15}$$

$$v_R = (r_x + 1)(r_y + 1) \tag{4.16}$$

However, more than one backward neighbor (BN) can share a segment with β_i. After an exhaustive case study of the overlapping regions among boxes in the CUDB model, where each possible neighborhood has been analyzed, it can be concluded that, there can be 1, 2 or 3 backward neighboring boxes that share a segment with β_i.

In the first case only the shared rectangle must be computed and subtracted. Note that in some cases a degenerated rectangle is obtained (see boxes β_i and β_k in Figure 4.19(d)). In the case of two BN of β_i sharing a segment S, it has been subtracted twice and therefore it must be added again. For example, in Figure 4.19(c) the red regions computed when analyzing the pairs of boxes (β_i, β_k) and (β_i, β_t) have been subtracted and therefore the yellow region has

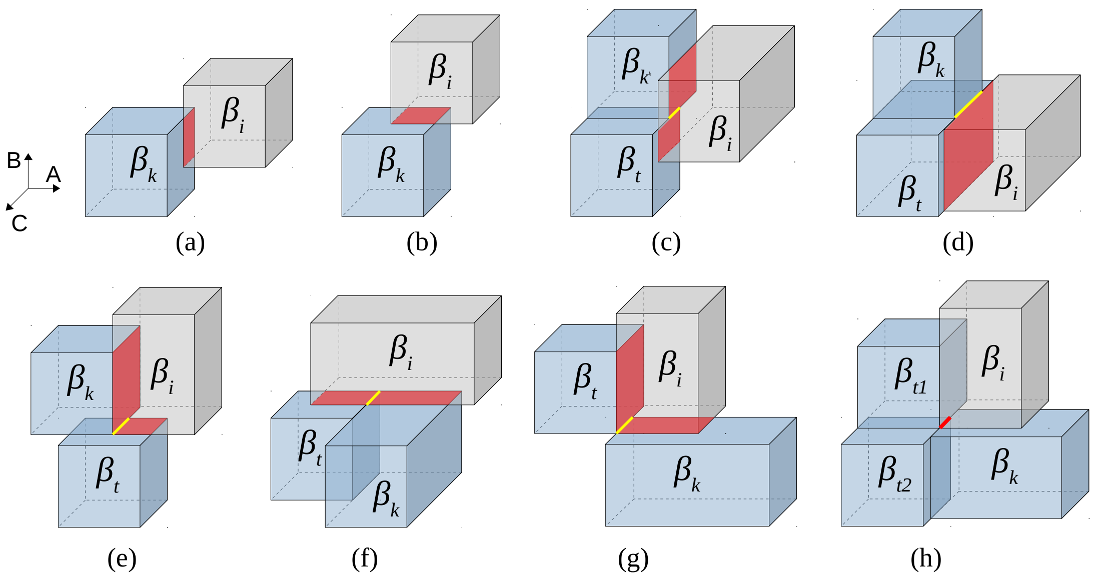


Figure 4.19: Backward neighbors configurations of a box β_i. (a) One ABN sharing a rectangle. (b) One BBN sharing a rectangle. (c) Two ABN sharing a segment. (d) degenerated case of (c). (e) One ABN and one BBN sharing a segment. (f) Two BBN sharing a segment. (g) One BBN and one ABN sharing a segment. (h) One BBN and two ABN sharing a segment.

been subtracted twice and has to be added again. There are 4 possible configurations for two BN of β_i sharing a segment, which are described below.

Configuration C1. Two boxes $\beta_k, \beta_t \in \beta_i.ABN$, where $\beta_t \in \beta_k.BBN$. See Figures 4.19(c) and (d).

Configuration C2. One box $\beta_k \in \beta_i.ABN$ and one box $\beta_t \in \beta_i.BBN$, where $\beta_t \in \beta_k.BBN$. See Figure 4.19(e).

Configuration C3. Two boxes $\beta_k, \beta_t \in \beta_i.BBN$, where $\beta_t \in \beta_k.ABN$. See Figure 4.19(f).

Configuration C4. One box $\beta_k \in \beta_i.BBN$ and one box $\beta_t \in \beta_i.ABN$, where $\beta_t \in \beta_k.ABN$. See Figure 4.19(g).

Therefore, let s be the length of the shared segment S, the number of enclosed unitary edges (e_S) and vertices (v_S) of S are computed as:

$$e_S = s \tag{4.17}$$

$$v_S = s + 1 \tag{4.18}$$

Finally, in the case of three BN of β_i sharing a segment S, there is only one possible configuration:

Configuration C5. One box $\beta_k \in \beta_i.BBN$ and two boxes $\beta_{t1}, \beta_{t2} \in \beta_i.ABN$, where $\beta_{t1}, \beta_{t2} \in \beta_k.ABN$ and $((\beta_{t1} \in \beta_{t2}.BBN) \wedge (\beta_{t2} \in \beta_{t1}.BBN))$. See Figure 4.19(h).

Note that configuration C5 is equivalent to two occurrences of C1 (($\beta_k, \beta_{t1}, \beta_{t2}$) and ($\beta_i, \beta_{t1}$, β_{t2})) and two occurrences of C4 (($\beta_i, \beta_k, \beta_{t1}$) and ($\beta_i, \beta_k, \beta_{t2}$)). However, both configurations C4 occur when β_i is being analyzed and, therefore, some shared elements with β_{t1} in one occurrence and with β_{t2} in the other, are added twice, so, the segment S shared by $\beta_i, \beta_k, \beta_{t1}$ and β_{t2} (highlighted in red in Figure 4.19(h)), represents the region that must be re-subtracted.

Algorithm 13: computeConnectivity

```
Input  : $Q$;                                                   /* ABC-sorted CUDB /*
Output: $\chi$, genus;
$n_0 \leftarrow 0$; $n_1 \leftarrow 0$; $n_2 \leftarrow 0$; $n_3 \leftarrow 0$;
for all box $\beta_i \in Q$ do
    Compute $\gamma_i$, $f_i$, $e_i$, $v_i$;
    for all $\beta_k \in \beta_i.ABN$ do
        Compute $f_R$, $e_R$ and $v_R$; $f_i \leftarrow f_i - f_R$; $e_i \leftarrow e_i - e_R$; $v_i \leftarrow v_i - v_R$;
        for all $\beta_t \in (\beta_i.ABN \cup \beta_i.BBN)$ do
            if $\beta_t \in \beta_k.BBN$ then                   /* Configurations C1 and C2 /*
                Compute $e_S$ and $v_S$; $e_i \leftarrow e_i + e_S$; $v_i \leftarrow v_i + v_S$;
            end if
        end for
    end for
    for all $\beta_k \in \beta_i.BBN$ do
        Compute $f_R$, $e_R$ and $v_R$; $f_i \leftarrow f_i - f_R$; $e_i \leftarrow e_i - e_R$; $v_i \leftarrow v_i - v_R$;
        for all $\beta_t \in \beta_i.BBN$ do
            if $\beta_t \in \beta_k.ABN$ then                   /* Configuration C3 /*
                Compute $e_S$ and $v_S$; $e_i \leftarrow e_i + e_S$; $v_i \leftarrow v_i + v_S$;
            end if
        end for
        $L \leftarrow \emptyset$;                               /* List of box pointers /*
        for all $\beta_t \in \beta_i.ABN$ do
            if $\beta_t \in \beta_k.ABN$ then                   /* Configuration C4 /*
                Compute $e_S$ and $v_S$; $e_i \leftarrow e_i + e_S$; $v_i \leftarrow v_i + v_S$;
                Insert $\beta_t$ to $L$;
            end if
        end for
        for each pair ($\beta_{t1}$,$\beta_{t2}$) : $\beta_{t1}, \beta_{t2} \in L$ do
            if ($\beta_{t1} \in \beta_{t2}.BBN$) or ($\beta_{t2} \in \beta_{t1}.BBN$) then   /* Configuration C5 /*
                Compute $e_S$ and $v_S$; $e_i \leftarrow e_i - e_S$; $v_i \leftarrow v_i - v_S$;
            end if
        end for
    end for
    $n_0 \leftarrow n_0 + v_i$; $n_1 \leftarrow n_1 + e_i$; $n_2 \leftarrow n_2 + f_i$; $n_3 \leftarrow n_3 + \gamma_i$;
end for
$\chi \leftarrow n_0 - n_1 + n_2 - n_3$;
$h_0 \leftarrow Q.CCL()$;
Compute the complement of $Q$ ($Q^c$) with 6-connectivity;
$h_2 \leftarrow Q^c.CCL() - 1$;
$genus \leftarrow h_0 + h_2 - \chi$;
```

The number of enclosed unitary edges (e_S) and vertices (v_S) are computed as in Expressions 4.17 and 4.18. This case is solved by inserting all the boxes β_t of configurations C4 into a list, then this list is analyzed in order to check for boxes that are neighbors in the B-direction.

Algorithm 13 shows the steps of the χ and the genus computation considering the (26, 6) adjacency pair and for any ABC-sorted CUDB. This algorithm computes the same number of voxels, faces, edges and vertices that the voxel-based method. The Euler characteristic, χ, is computing using Expression 4.7. To compute the number of connected components and internal cavities, the connected components of the object's complement, the CUDB-CCL process is applied. Finally, the genus is computed using Expression 4.8.

4.3.3 Connectivity Results

The Euler characteristic and the genus has been computed for a selection of datasets with different shape features and size. They present non-manifold configurations and may contain isolated cavities and disconnected components. Figure 4.20 shows rendered views of the test datasets and its size in the voxel model. All datasets come from public volume repositories where from (i) to (o) are real volume data coming from CT or MRI scanners. Three methods have been compared: a voxel-based [90], an EVM-based [13] (see Section 2.3.3) and the CUDB-based, presented in this thesis. These methods produce exactly the same results. The

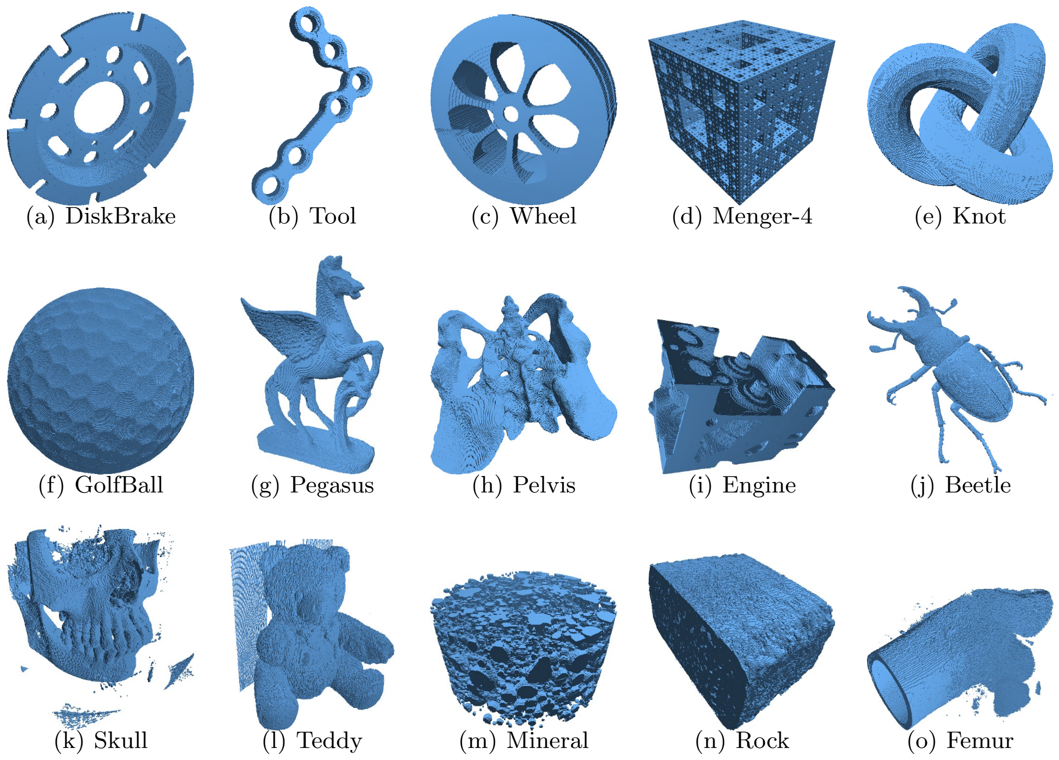


Figure 4.20: Rendered images of the test datasets.

corresponding programs have been written in C++ and tested on a PC Intel®Core 2 Duo CPU E6600@2.40GHz with 3.2 GB RAM and running Linux.

As the CUDB is obtained from the EVM and, in turn, EVM can be obtained from the voxel model, we consider that these models are available and ignore the cost of conversion from the voxel model, similar to the previous EVM-based method. Therefore, EVM is used o compute the complement of the input object representation, whose run time is negligible. However, the conversion time to compute the CUDB-representation of the complement, is considered in the computation time of the CUDB-based method.

Table 4.2 shows the attributes of the tested datasets and Table 4.3 shows the required time in seconds to compute χ and the genus for each referenced method. The last columns of this latter table show the conversion times for voxel model to EVM and EVM to CUDB of the original object, in order to show that even considering these costs, the overall time of the CUDB-based method is better than the voxel-based.

To compute the genus, the three methods need to compute the complement of the object, and besides, the EVM-based method needs to convert the object to a homotopic manifold analog. Note that the CUDB-based method is very fast to compute χ, and regarding the genus computation, it is by far, faster than the voxel-based method, in some datasets up to two orders of magnitude (GolfBall, Pegasus, Pelvis, and Beetle). Compared with the previous EVM-based method the CUDB-based is also faster in all the tested datasets, in some of them up to an order of magnitude (Pegasus, Pelvis, Teddy and Femur). Furthermore, note that even considering the conversion time $E \to C$, the CUDB-based method remains faster than EVM-based.

Table 4.2: Attributes of the test datasets. For each dataset: size in voxels, number of foreground voxels ($|FV|$), number of boxes in CUDB ($|CUDB|$). Next, with an adjacency pair (26, 6): number of connected components (C^+), number of isolated cavities (C^-), χ and genus.

Dataset	size	$\|FV\|$	$\|CUDB\|$	C^+	C^-	χ	genus
DiskBrake	299×300×43	528798	10310	1	0	-20	11
Tool	511×339×48	1778611	11000	1	0	-10	6
Wheel	120×300×300	3809958	13207	1	0	-14	8
Menger-4	162×162×162	1280000	46704	1	0	-52864	26433
Knot	329×350×257	7509337	76831	1	0	0	1
GolfBall	510×509×511	13645424	129493	1	0	2	0
Pegasus	598×800×574	24683709	191747	1	8	2	8
Pelvis	905×1259×1108	88338560	506000	1	8	-8	13
Engine	139×197×108	901818	25524	9	194	146	130
Beetle	411×371×247	1737343	36052	17	114	-190	226
Skull	256×256×256	1112906	114563	1624	337	-1020	2471
Teddy	424×321×493	24758866	124063	59	212	-2580	1561
Mineral	376×375×206	7363953	232008	724	5	-2792	2125
Rock	240×406×267	19348939	331491	1336	17263	25828	5685
Femur	463×494×628	4014089	838585	22714	7909	-25144	43195

Table 4.3: Run time comparison in seconds. For each dataset the time to compute χ and the genus for each referenced method, voxel-based (*mvx*), EVM-based (*evm*) and CUDB-based (*cudb*). Next, the times for voxel model to EVM ($V \to E$) and EVM to CUDB ($E \to C$) conversion. The last column represents $T = V \to E + E \to C + cudb^*$.

Dataset	Time χ			Total time (genus)			$V \to E$	$E \to C$	T
	mvx	*evm*	*cudb*	*mvx*	*evm*	*cudb**			
DiskBr.	0.66	0.28	**0.01**	2.51	0.81	**0.10**	0.78	0.12	*1.00*
Tool	1.77	0.26	**0.01**	6.22	0.84	**0.11**	1.47	0.13	*1.71*
Wheel	3.84	0.24	**0.01**	10.84	1.47	**0.20**	2.60	0.20	*3.00*
Menger-4	1.30	0.70	**0.02**	3.85	1.52	**0.27**	1.05	0.21	*1.53*
Knot	9.19	1.94	**0.05**	29.07	5.83	**0.64**	1.94	0.49	*3.06*
GolfBall	41.53	2.90	**0.07**	150.37	9.49	**1.20**	34.72	0.89	*36.81*
Pegasus	71.54	6.15	**0.13**	274.83	20.81	**1.88**	72.82	1.51	*76.21*
Pelvis	354.65	13.21	**0.33**	1338.77	43.98	**4.27**	23.33	0.55	*24.59*
Engine	0.77	0.62	**0.02**	2.43	1.77	**0.21**	0.71	0.21	*1.12*
Beetle	11.08	2.01	**0.02**	39.39	2.34	**0.29**	9.49	0.25	*10.03*
Skull	5.74	3.51	**0.06**	19.40	9.16	**0.94**	5.10	0.74	*6.77*
Teddy	20.97	3.49	**0.09**	74.07	10.39	**1.02**	18.23	0.77	*20.01*
Mineral	9.35	5.92	**0.16**	31.04	19.25	**1.97**	8.76	1.59	*12.32*
Rock	12.48	9.47	**0.39**	33.89	24.43	**3.44**	8.65	2.06	*14.14*
Femur	45.94	31.16	**0.80**	170.64	80.66	**8.06**	41.90	5.69	*55.65*

4.4 Sphericity and Roundness

In this section, methods to compute the sphericity and roundness of binary volume datasets are presented. In order to compute the three representative axes of the object, its oriented bounding box (OBB) is obtained first (see Section 4.4.1). Then, several sphericity indices can be computed (see Section 4.4.2).

Additionally, a 3D roundness index is proposed. Section 4.4.3 presents a ray-casting-like approach based on EVM that uses basic geometric methods such as a transformation matrix and the ray-ellipsoid intersection.

4.4.1 Oriented Bounding Box Computation

An OBB is a box which may be arbitrarily oriented, whose faces have normals which are pairwise orthogonal. An OBB can be represented with a center point $\mathbf{c}$, three edge half-lengths h_1, h_2 and h_3, and an orientation specified with three mutually orthogonal unit vectors $\mathbf{v}^1$, $\mathbf{v}^2$ and $\mathbf{v}^3$ (see Figure 4.21):

$$OBB = \left\{ \mathbf{c} + \sum_{i=1}^{3} x_i h_i \mathbf{v}^i \ : \ x_i \in [-1, 1] \right\} \tag{4.19}$$

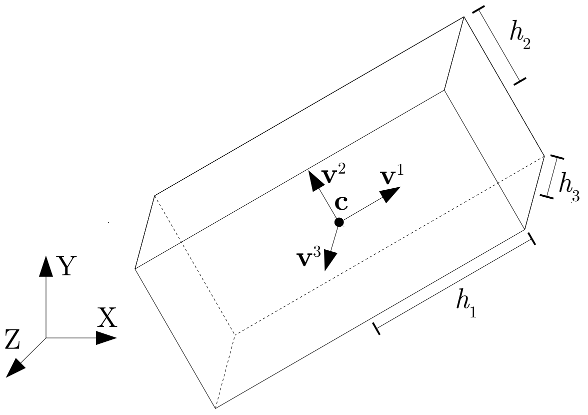


Figure 4.21: A 3D arbitrarily oriented bounding box.

An OBB can be constructed by examining the object's point set. This point set forms a cloud and have some statistical distribution characterized by a mean $\mathbf{m} = (m_1, m_2, m_3)$, and a covariance matrix $\mathbf{C}$. The mean describes the center of mass and the covariance matrix contains information about how the cloud is approximately spread out. Eigenvectors of that matrix give the orientation along which the cloud has maximum and minimum statistical spread [47].

Given a set of n 3D points, viewed as vectors whose initial point is the origin: $\mathbf{p}^1, \mathbf{p}^2, \ldots, \mathbf{p}^n$, such that $\mathbf{p}^k = (p_1^k, p_2^k, p_3^k)$, the mean is defined as:

$$m_i = \frac{1}{n} \sum_{k=1}^{n} p_i^k, \quad i = 1, 2, 3 \tag{4.20}$$

and the 3×3 covariance matrix **C** is defined as:

$$\mathbf{C}_{ij} = \mathrm{Cov}[i,j] = \left(\frac{1}{n} \sum_{k=1}^{n} p_i^k p_j^k \right) - m_i m_j, \quad i,j = 1,2,3 \tag{4.21}$$

The normalized eigenvectors of matrix **C** represent the orientation vectors of the OBB. Let $\mathbf{v}^1$, $\mathbf{v}^2$ and $\mathbf{v}^3$ be these eigenvectors. The lower (L) and upper (U) extremes along each axis are given by projecting all points $\mathbf{p}^k$ onto each eigenvector and checking the minimum and maximum coordinates in each direction:

$$L = \{l_1, l_2, l_3\} = \left\{ min(\mathbf{v}^1 \cdot \mathbf{p}^k),\ min(\mathbf{v}^2 \cdot \mathbf{p}^k),\ min(\mathbf{v}^3 \cdot \mathbf{p}^k) \right\}, \quad \forall\, k = 1, \ldots, n \tag{4.22a}$$

$$U = \{u_1, u_2, u_3\} = \left\{ max(\mathbf{v}^1 \cdot \mathbf{p}^k),\ max(\mathbf{v}^2 \cdot \mathbf{p}^k),\ max(\mathbf{v}^3 \cdot \mathbf{p}^k) \right\}, \quad \forall\, k = 1, \ldots, n \tag{4.22b}$$

As the i-th axis of the OBB is aligned with $\mathbf{v}^i$, the OBB's edge half-length h_i along this axis is given by:

$$h_i = \frac{u_i - l_i}{2}, \quad i = 1,2,3 \tag{4.23}$$

and the center point **c** of the OBB is given by:

$$\mathbf{c} = \frac{1}{2} \sum_{i=1}^{3} (l_i + u_i) \mathbf{v}^i \tag{4.24}$$

The same computed parameters define also the ellipsoid inscribed into the OBB, whose principal axes have lengths $a = 2h_1$, $b = 2h_2$, and $c = 2h_3$. We prefer to represent the ellipsoid with the full-axes length instead of the semi-axes length in order to be consistent with the equations used to compute the sphericity and roundness indices (see Section 2.3.5).

The OBB produced with the covariance method may not be a tight fit if the points are not well distributed. The reason for this is that variations in point sampling density can skew the eigenvectors of the covariance matrix and result in a poor orientation. However, as the set of points to be evaluated comes from solid samples represented with a voxel model, a good distribution can be expected.

The covariance matrix can be built from the corresponding voxel model using the voxel coordinates as representative points. However, the extreme vertices (EV) of the EVM-represented object can be used as the set of representative points. Although the number of EV is significantly smaller than the number of voxels, the OBB produced from the EVM is very similar to the OBB produced from the voxel model. From the analysis of the test samples, the maximum volume difference computed between a voxel-based and a EVM-based OBB was less than 8.5%, while the computation time is greatly reduced in the EVM-based method. Figure 4.22 shows some samples and their computed OBB.

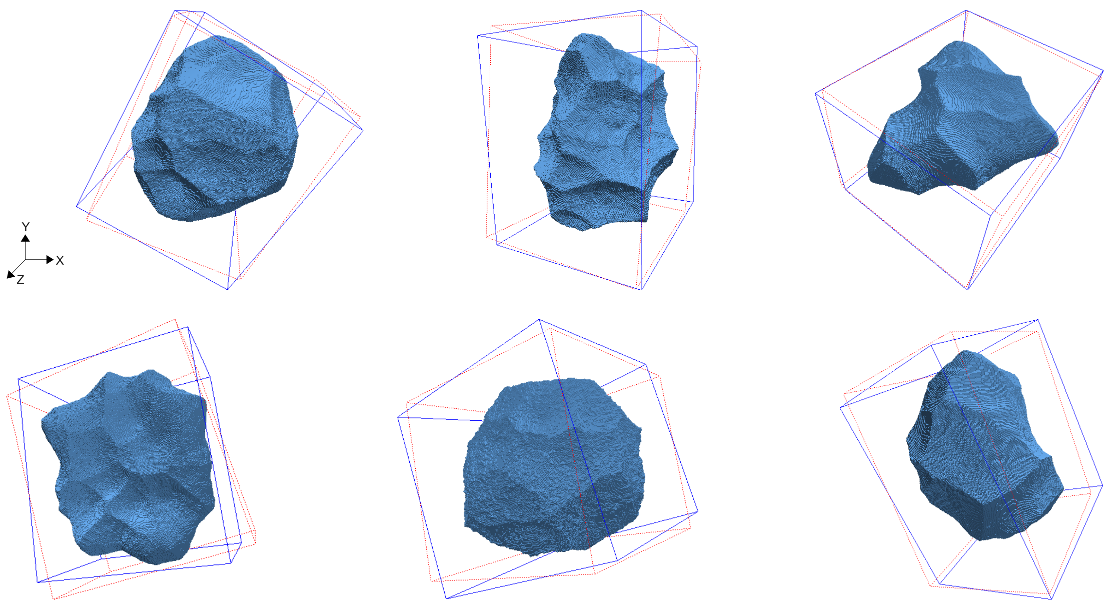


Figure 4.22: Four rock samples and their OBB. The EVM-based and voxel-based OBB in continuous (blue) and stippled (red) line respectively.

4.4.2 Sphericity Computation

Given an input object, the OBB is computed first. Then, the sphericity indices given by Equation 2.16 to 2.19 can be directly computed from the three principal axes lengths (a, b, c) of the inscribed ellipsoid. To compute the true sphericity index (Equation 2.15), the surface area of the object is required. Because of the nature of EVM, the surface area of an EVM-represented object is measured with a block-form surface extraction algorithm [127] and therefore, this method is not suitable to estimate the object's continuous surface area. However, there are voxel-based methods that better estimate this value for binary volumes.

The surface area is estimated using a voxel-based scheme [97, 181], which is unbiased for random plane orientations with small error when applied to curved surfaces. The algorithm begins by detecting all the surface voxels (voxels with 6-connectivity to background voxels), then, these voxels are classified into nine classes (denoted S_1 to S_9) according to the number and configuration of its faces that are exposed to the background (see Fig. 4.23). Then, the surface area $\mathcal{S}$, is estimated as a linear combination of the class membership values N_i.

$$\mathcal{S} = \sum_{i=1}^{9} W_i N_i \tag{4.25}$$

The optimal computed weights W_i associated with the voxels in classes S_i are: $W_1 \approx 0.894$, $W_2 \approx 1.3409$, $W_3 \approx 1.5879$, $W_4 = 2$, $W_5 = \frac{8}{3}$, $W_6 = \frac{10}{3}$ [97], $W_7 \approx 1.79$, $W_8 \approx 2.68$, $W_9 \approx 4.08$ [181].

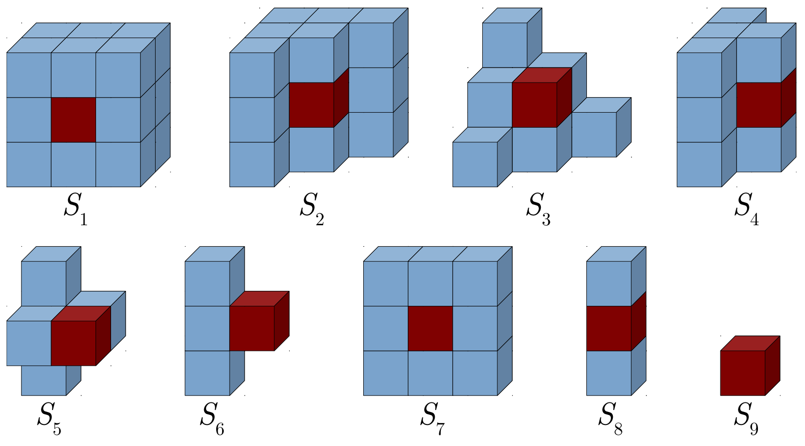


Figure 4.23: The nine unique surface voxel classes (modulo reflections and rotations) [181].

4.4.3 Roundness Computation

In industrial engineering, the roundness measurement of workpieces is estimated by doing a single 2D trace covering 360°of the workpiece, this process is usually performed with a turntable-type instrument or a stylus-type instrument [122]. The deviation of the trace from a least-squares circle fit to the data at equally spaced angles θ_i gives a roundness estimation of $d_i - r$, where r is the radius of the circle and d_i the distance from the circle center to the trace [99]. A set of random, uniformly distributed, rays can be traced to the surface of a reference ellipsoid [156].

Based on this previous idea, a 3D roundness index is proposed. The deviations of the rays from the surface of the object gives a roundness estimation. For efficiency purposes, instead of generating a set of uniformly distributed rays, we propose to trace rays to each EV in the EVM-represented object. Then, to compute the proposed EVM-roundness index, three steps must be performed:

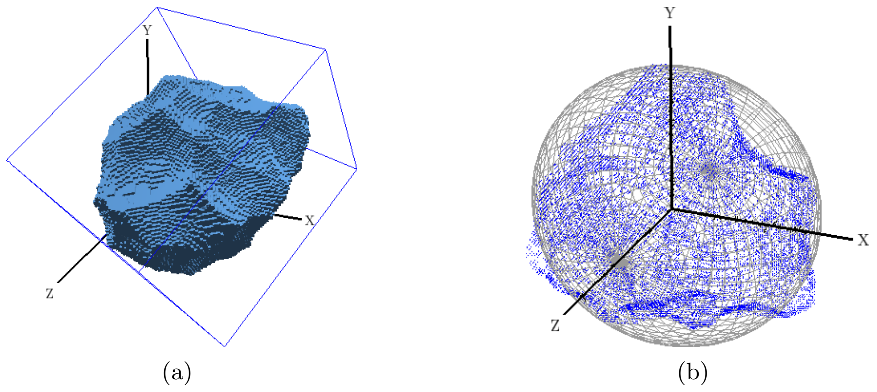
1. Compute the OBB of the object to obtain the principal axes of the inscribed ellipsoid.

2. Create a transformation matrix $\mathbf{M}$, which transforms the OBB so that it is aligned to the main coordinate axes and centered at the origin in order to facilitate subsequent computations. Apply this transformation to all the EV of the object.

3. Trace a ray from the origin to each EV representing point $\mathbf{p}^k$ of the object, and measure the distance Δ_k, between $\mathbf{p}^k$ and the point $\mathbf{q}^k$, where the ray intersects the surface of the ellipsoid.

The first step computes the OBB with the method described in Section 4.4.1. See Figure 4.24(a).

The second steps creates a transformation matrix $\mathbf{M}$, which is a composition of a translation matrix that displaces the ellipsoid center to the origin and three rotation matrices necessary to align the ellipsoid principal axes to the main coordinate axes. Figure 4.24(b) shows the EV set



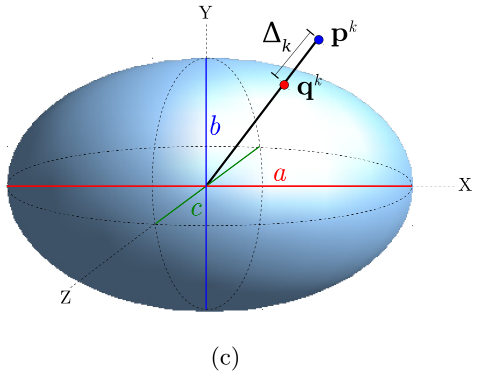


Figure 4.24: Steps to compute the EVM-roundness. (a) Calculate the OBB. (b) Transform the model such that the reference ellipsoid is in the origin. (c) Trace rays to each EV $\mathbf{p}^k$ and compute Δ_k.

of an object and the reference ellipsoid after transformation. Note that the OBB bounds both the object and the inscribed ellipsoid, but the object and the ellipsoid may intersect each other.

The third step applies the algorithm for ray-sphere intersection [146] properly adjusted for an ellipsoid (see Figure 4.24(c)). The ray has an equation of the form $\mathbf{q} = \mathbf{p_0} + t\mathbf{p}$. Let $\mathbf{p_0}$ be the origin $(0, 0, 0)$, the point $\mathbf{q} = (q_1, q_2, q_3)$ where the ray, passing by the EV point $\mathbf{p} = (p_1, p_2, p_3)$, crosses the ellipsoid with center at $\mathbf{p_0}$ and principal axes lengths a, b and c, can be computed solving the next quadratic equation for t:

$$t^2|\mathbf{p}'|^2 - 1 = 0, \quad \text{where } \mathbf{p}' = \left(\frac{2p_1}{a}, \frac{2p_2}{b}, \frac{2p_3}{c}\right) \tag{4.26}$$

therefore,

$$\mathbf{q} = (tp_1, tp_2, tp_3) \tag{4.27}$$

$$\Delta = |\mathbf{p} - \mathbf{q}| \tag{4.28}$$

The EVM-roundness index is defined as the average of the differences Δ_k for all the EV representing points $\mathbf{p}^k$. As the measurements depend on the size of the input object, to remove the size effect, the average is divided by a length factor. There are several alternatives for this factor, e.g., the principal axes lengths (a, b, or c), the geometric mean of them or the corresponding arithmetic mean. In the presented method the geometric mean of a, b, and c is adopted as length factor since the EVM-roundness index is compared with a method that uses this length factor. The average is also multiplied by a factor of 10 to enhance the readability of the results.

$$\mathcal{R} = \frac{10}{n(abc)^{\frac{1}{3}}} \sum_{k=1}^{n} \Delta_k \tag{4.29}$$

where n is the number of extreme vertices.

4.4.4 EVM-roundness Correlation

In order to show the correlation of the EVM-roundness index with the roundness index defined by Wadell, the silhouettes of the Krumbein's chart (see Figure 2.10) have been tested in a 2D version of the proposed method. Each tested image was created with a resolution of $\approx 320^2$ pixels. Figure 4.25 shows the relationship between Krumbein's roundness and EVM-roundness. These results have a linear correlation of -0.898 (negative as Krumbein's roundness index decreases while EVM-roundness index increases). The EVM-roundness index also has been computed applying the OBB computed from the voxel model. It results in a better, but not very different correlation of -0.902.

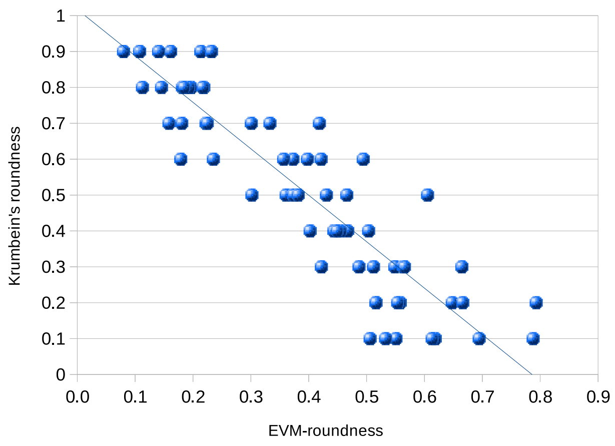


Figure 4.25: Relationship between Krumbein's chart roundness and EVM-roundness.

For a comparison in 3D, the roundness index proposed by Hayakawa and Oguchi (HO-roundness) $\mathcal{R} = \frac{\mathcal{V}}{\mathcal{S}(abc)^{\frac{1}{3}}}$ [58] has been also computed. To get the HO-roundness, the surface area of the object is measured using the voxel-based scheme described in Section 4.4.2.

We have used a GPL Blender extension, `rockGenerator`, to create a set of thirty 3D models of rocks (see Figure 4.26). Each model was converted to a voxel model with a resolution of $\approx 250^3$ voxels. Distribution of the set of rocks according to their computed sphericity and roundness is shown in Figure 4.27. The relationship between HO-roundness and EVM-

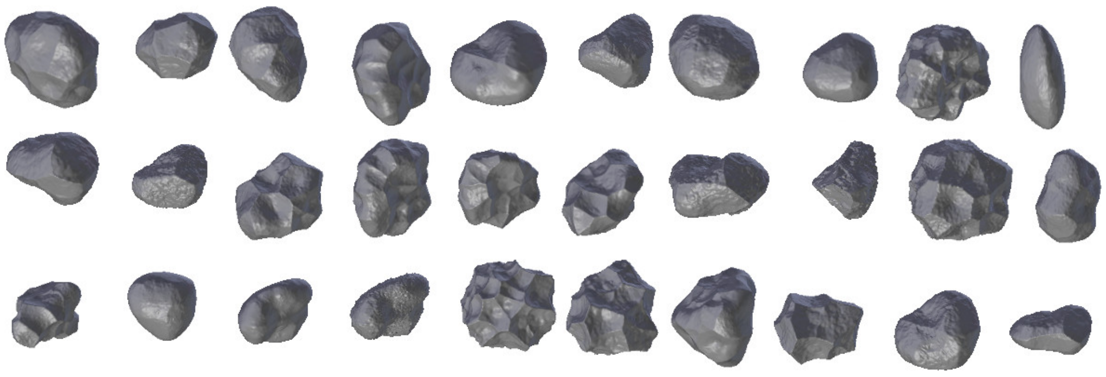


Figure 4.26: Thirty 3D rock samples created with rockGenerator.

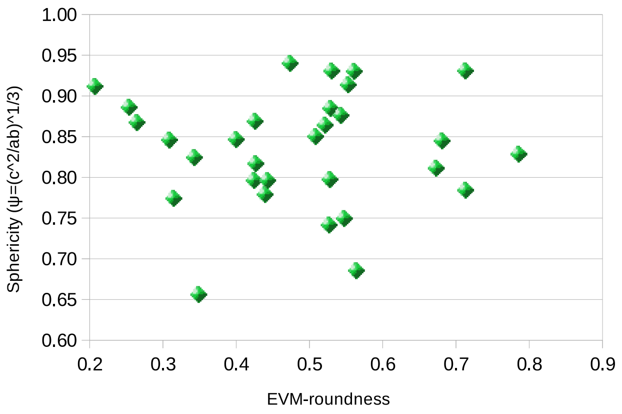


Figure 4.27: Distribution of the rocks samples according to their sphericity and roundness.

roundness indices is shown in Figure 4.28. In this case, the results have a correlation of -0.938.

The corresponding programs have been written in C++ and tested on a PC Intel®Core 2 Duo CPU E6600@2.40GHz with 3.2 GB RAM and running Linux. In this PC, the time to compute the HO-roundness index for each rock sample is about 2 seconds, while for the EVM-roundness index, it is about 0.1 seconds.

4.5 Conclusions

This chapter has presented new methods to compute the porosimetry, connectivity sphericity and roundness of binary volume datasets. Next, some observations related to the proposed methods are presented.

The CUDB-based virtual porosimeter simulates mercury intrusion in a porous medium and does not require prior computation of the skeleton. As the prior computation of the skeleton is very time-consuming, the presented approach achieves a noticeable reduction in time. The key feature of this approach is the throat computation, because these throats make possible to

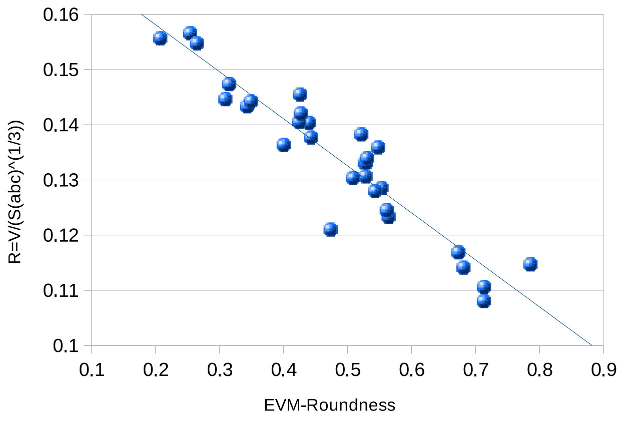


Figure 4.28: Relationship between HO-roundness and EVM-roundness.

separate regions corresponding to different diameters. A flood-fill process is then applied using a CCL algorithm to the reduced number of boxes of the CUDB model. The *in silico* approaches are based on the geometry of the analyzed samples and the results obtained are the expected theoretical results as shown in the experimental results section. However, comparing the results of physical and in-silice methodologies is still a challenging task.

The method to compute the Euler characteristic and the genus exploits the advantages of CUDB. It can be concluded that computing the connectivity is notably faster in the proposed approach. The performance variability is caused by the dataset size but above all to their surface intricacy: the voxel-based method performance is function of the number of voxels, but the proposed method depends on the number of boxes, tightly related to the model's tortuosity (a property that represents the twist of a curve, i.e., the degree of turns or detours a model has [52]), like the EVM-based method.

Finally, the method to estimate the sphericity is based on the computation of the object's OBB, which can be computed in a faster way directly from EVM. From the three OBB edge lengths, several sphericity indices can be computed, including the true sphericity index where a voxel-based scheme is applied to estimate the real surface area of curved surfaces. Regarding the roundness, a new EVM-based roundness index has been proposed. The method is based on the trace of rays to the vertices of the EVM-represented object and their intersections with a reference ellipsoid. The resulting roundness index shows a good correlation with the Krumbein's chart and a better correlation with a previous 3D roundness index. Besides, the time to compute the proposed index is very fast compared with previous voxel-based and manual methods.

CHAPTER 5

Lossless Orthogonal Simplification

5.1 Introduction

In this chapter, a new approach to simplify objects is presented. It is restricted to orthogonal pseudo-polyhedra (OPP). From an initial OPP, the method computes a level-of-detail (LOD) sequence of bounding volumes that are also OPP. The sequence is finite and can end with a convex OPP that is the AABB of all the sequence. Such bounding structures are denoted as BOPP (bounding OPP). BOPP satisfy basically two properties: (1) any BOPP contains the previous one and, therefore, all of them completely contain the original object, (2) all of the BOPP, as well as the original object, have the same AABB.

OPP are represented with EVM and the simplification is achieved using a new approach called *merging faces* (see Section 5.2), which displaces sets of faces and merges them by applying 2D Boolean operations. The method deals with general 3D orthogonal objects with any number of shells, cavities and through holes. Different suitable strategies for merging faces are analyzed, and a technique that relies on the object continuity in order to get a better shape preservation by avoiding abrupt changes is also devised.

5.2 Merging Faces Strategy

5.2.1 Basic merging

The idea of the *merging faces* strategy is to work with pairs of consecutive cuts of an OPP, where for each cut of a pair, a displacement of its faces in the direction of their corresponding normal vector is applied and, then, the displaced faces are merged with those faces of the other cut with the same normal vector. We define as $B(P)$, the bounding OPP (BOPP) of an OPP P obtained with the *merging faces* strategy.

Formally, let P be an OPP and let C_A and C_B be two consecutive cuts of P and FD_A, BD_A, FD_B and BD_B their corresponding forward and backward differences (see Equation 2.2.4). To obtain a coarsened OPP, the *merging faces* process displaces BD_B to the position of BD_A and displaces FD_A to the position of FD_B. Then, the new cuts $newC_A$ and $newC_B$

that represent the merged faces and replace C_A and C_B respectively in the input object, are computed according to the next equation:

$$\overline{newC_A} = \overline{BD_A} \cup^* \overline{BD_B} \tag{5.1a}$$

$$\overline{newC_B} = \overline{FD_A} \cup^* \overline{FD_B} \tag{5.1b}$$

The merging process displaces faces in the direction of their respective normal vector, i.e., outward of the object, but only those faces in which this displacement goes to the consecutive cut (faces FD_A of C_A and BD_B of C_B) and then performs the union of these displaced faces with the remaining faces of each cut which have the same normal vector (FD_B of C_B and BD_A of C_A, respectively). Figure 5.1 shows a simple 2D example.

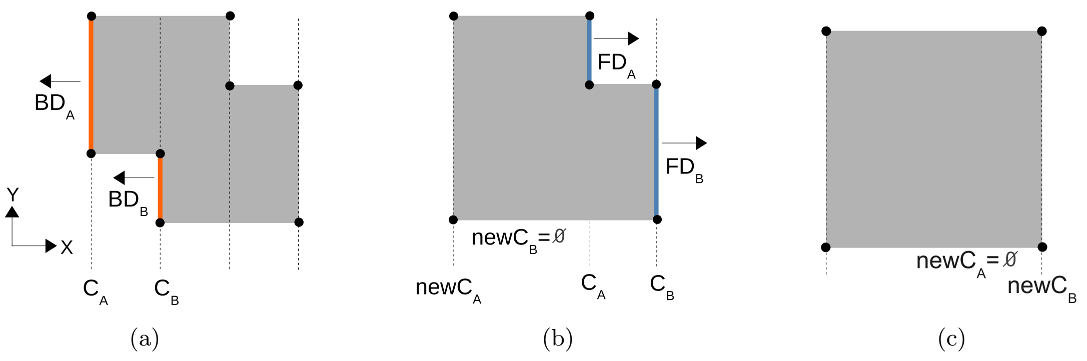


Figure 5.1: 2D example of *merging faces*: (a) In a first step, BD_B is displaced to the position of C_A. (b) The result of the first step is $newC_A$ obtained by merging BD_B and BD_A and $newC_B = \emptyset$, then, in a second step, FD_A is displaced to the position of C_B. (c) The resulting BOPP is obtained by merging FD_A and FD_B; now $newC_A = \emptyset$.

If P is represented as an ABC-sorted EVM, for a single traversal of cuts the application of this process only will coarsen P in the A-coordinate. Therefore, to obtain $B(P)$ the process must be repeated for the other two main directions. Figure 5.2 shows a 2D example where the process is applied for the two main directions. Observe that the result can be slightly different depending on the ordering in which the three main directions are selected. In order to speed up the computation, the best first candidate would be the ABC-ordering with A-axis having less number of cuts, but as EVM does not report this value, it can be approximated by $c_n - c_1$, c_1 and c_n being respectively the coordinates of the first and last cut in this direction.

Although non-extreme vertices are not stored in EVM, when cuts are decomposed in forward and backward differences, they appear (see Section 2.2.4) and the *merging faces* strategy deals correctly with them. For instance, Figure 5.2(a) shows a non-extreme vertex in C_B that shows up when BD_B and FD_B are computed and that appears in $newC_B$ (Figure 5.2(b)).

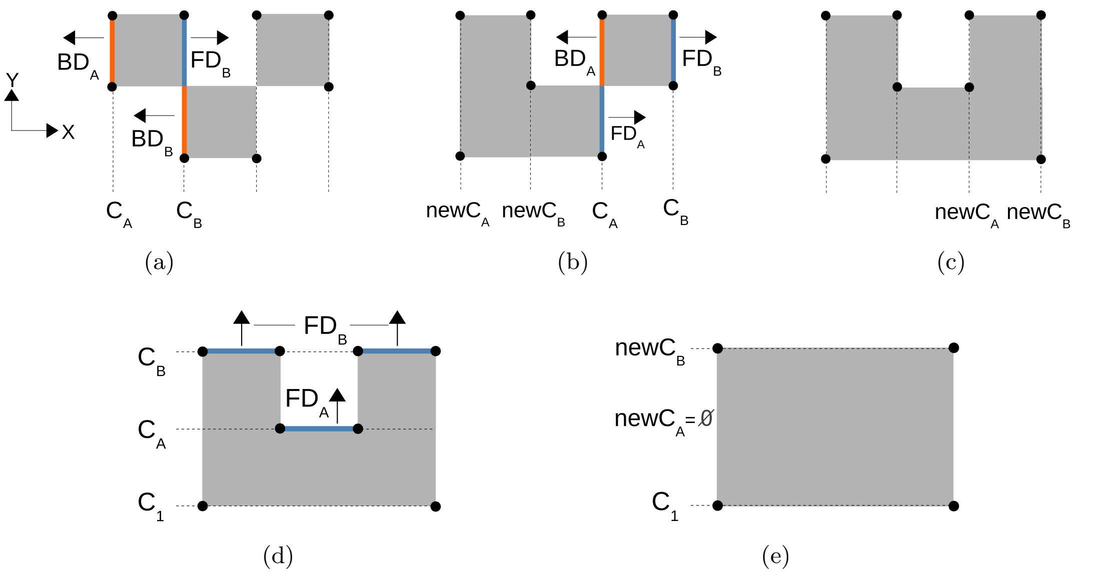


Figure 5.2: 2D example with non-EV vertices. (a) Faces displaced in a first step. (b) Result after applying Equation 5.1 and faces displaced in a second step. (c) Final result for the XY-sorting. (d) Faces displaced for the YX-sorting. (e) Resulting BOPP.

5.2.2 Treatment of the Void Space

An OPP may have any number of rings on faces, through holes and shells (cavities or connected components). In a general OPP, Equations 5.1 do not give the desirable result. For example, applying this equation to the pair of cuts depicted in Figure 5.3(a), the same OPP without coarsening is obtained. Notice that a hole in 2D is equivalent to a cavity in 3D. To deal with this issue, the void spaces are detected first and then closed.

Given the pair of consecutive cuts (C_A,C_B), BD_A and BD_B are sets of faces whose normal vectors point to the opposite direction of FD_A and FD_B, but only the pair formed by FD_A and BD_B can represent a void space between the cuts. Therefore, a void space is given by the

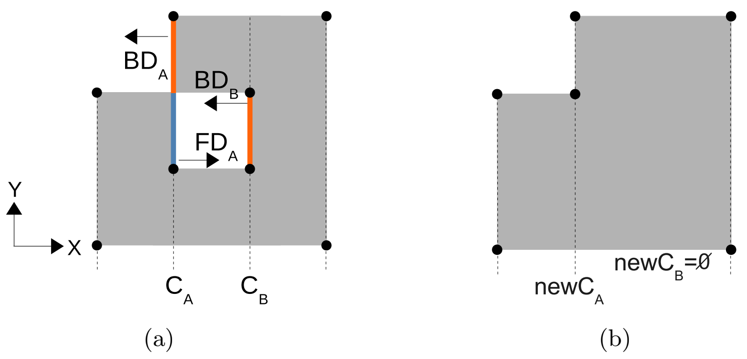


Figure 5.3: A 2D OPP with a single hole. (a) Faces merged in order to close the hole. (b)Result and end for the XY-sorting.

intersection of their projections, i.e.:

$$vSpace(C_A, C_B) = \overline{FD_A} \cap^* \overline{BD_B} \tag{5.2}$$

The removal of $vSpace$ from both $newC_A$ and $newC_B$, closes the void space between C_A and C_B. Then, Equations 5.1 are extended with Equation 5.2:

$$\overline{newC_A} = (\overline{BD_A} \cup^* \overline{BD_B}) -^* (\overline{FD_A} \cap^* \overline{BD_B}) \tag{5.3a}$$

$$\overline{newC_B} = (\overline{FD_A} \cup^* \overline{FD_B}) -^* (\overline{FD_A} \cap^* \overline{BD_B}) \tag{5.3b}$$

When $vSpace = \emptyset$ Equations 5.3 and 5.1 are equivalent. As this last operation contributes just to close interior void spaces, the subset property is still guaranteed. Figure 5.3 shows an example with a single hole. In Figure 5.3(a) $\overline{FD_A} = \overline{BD_B}$ and $vSpace(C_A, C_B) = \overline{FD_A} = \overline{BD_B}$. Then, applying Equation 5.3, $newC_B = \emptyset$ and the hole is closed (Figure 5.3(b)). As shows Equations 5.3, basic merging and void space removal are simultaneously performed. Note that the simple removal of the void space would have the same effect that the closing morphological operation.

In 3D, all types of void space between cuts are detected as those shown in Figure 5.4. There are cases in which the void space is not detected in all the three directions, but that is always detected in one or two of them. For example, in Figures 5.4(a) and 5.4(b) the void space is not detected in the Y-direction but is detected either in the X or Z-direction. However, there are other cases in which the void space is detected in all directions. In Figure 5.4(c) a void space is detected in the X or Y-direction and the through hole is partially closed, i.e., only the space

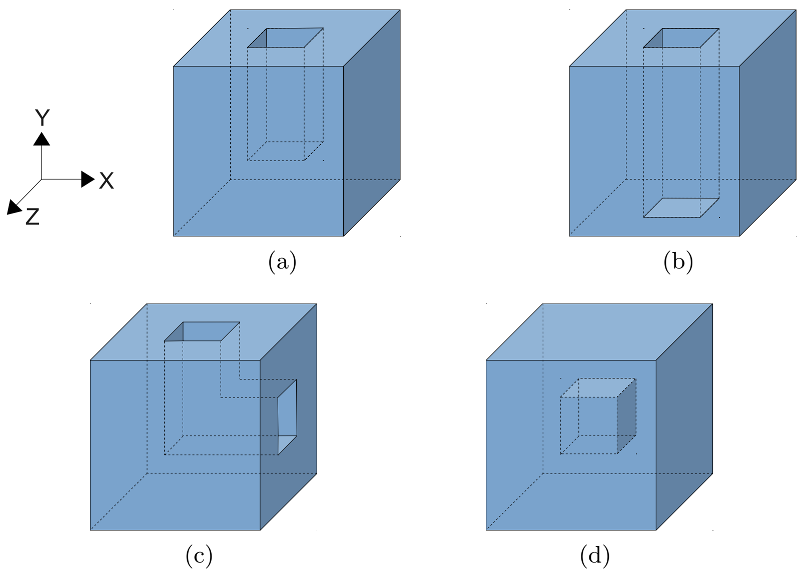


Figure 5.4: Some configurations of void spaces in 3D. (a) Concavity. (b) Through hole. (c) Through hole in L-shape, (d) Internal hole or cavity.

between the processed cuts gets closed, and the whole through hole will be closed in the next direction; but, if we begin with the Z-direction, the detected void space allows us to close the whole through hole. The cavity depicted in Figure 5.4(d) will be closed in any direction.

Although there are cases, as the simple concavity in Figure 5.4(a), that could be solved with the application of Equations 5.1, for general void spaces, as those in Figures 5.4(b), 5.4(c) and 5.4(d), Equations 5.3 must be applied in order to completely close the hole.

Figure 5.5(a) shows a 3D example with through holes, where applying Equations 5.3 in the X-direction, the computed new cuts, $newC_A = \emptyset$ and $newC_B$ (Figure 5.5(c)) partially close the hole, resulting in an object with a concavity (Figure 5.5(d)), which will be closed in the next direction.

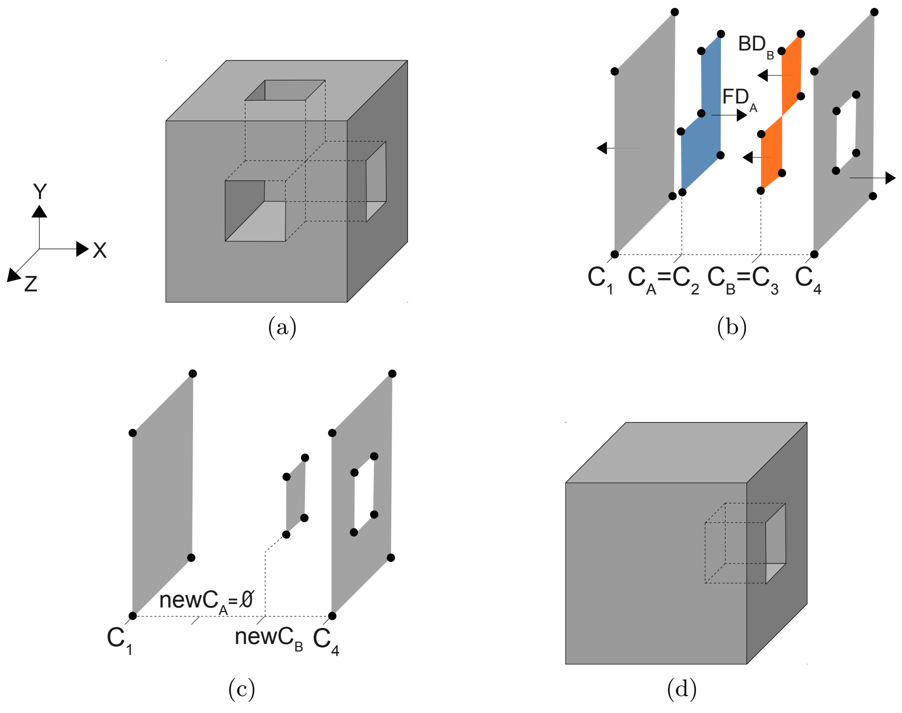


Figure 5.5: A 3D OPP with holes. (a) Original object. (b) Faces merged in a first step. (c) Resulting cuts and (d) Result for the XYZ-sorting.

5.2.3 Merging Faces Optimization

Next, two theorems that permit to rewrite Equation 5.3 in order to be faster to compute are proved.

Theorem 1. *The projection of FD and BD of two consecutive cuts (C_A,C_B) are quasi-disjoint sets respectively, i.e.,* $\overline{FD_A} \cap^* \overline{FD_B} = \emptyset$ *and* $\overline{BD_A} \cap^* \overline{BD_B} = \emptyset$.

Proof. The proof is based on the Jordan theorem and the fact that any ray crossing the boundary of the polyhedron, alternatively goes from outside to inside and vice versa. Therefore, in any OPP, assuming that $\overline{FD_A} \cap^* \overline{FD_B} \neq \emptyset$, would mean that a ray could cross FD_A going outside and then cross FD_B going outside again, which is a contradiction. The same reasoning applies to BD. □

Theorem 2. *$vSpace(C_A, C_B) \subseteq (\overline{BD_A} \cup^* \overline{BD_B})$ and $vSpace(C_A, C_B) \subseteq (\overline{FD_A} \cup^* \overline{FD_B})$.*

Proof. From Eq. 5.2, $vSpace(C_A, C_B) = \overline{FD_A} \cap^* \overline{BD_B}$. Without loss of generality:
$(\overline{FD_A} \cap^* \overline{BD_B}) \subseteq \overline{BD_B} \subseteq (\overline{BD_A} \cup^* \overline{BD_B})$, and thus,
$(\overline{FD_A} \cap^* \overline{BD_B}) \subseteq (\overline{BD_A} \cup^* \overline{BD_B})$
In a similar way $vSpace(C_A, C_B) \subseteq (\overline{FD_A} \cup^* \overline{FD_B})$ can be proved. □

Therefore, according to these theorems and EVM Properties 5 and 6 that permit to perform unions and differences in some special cases as simple point-wise XOR operations (see Section 2.2.4), Equations 5.3 can be rewritten as:

$$\overline{newC_A} = \overline{BD_A} \otimes^* \overline{BD_B} \otimes^* (\overline{FD_A} \cap^* \overline{BD_B}) \tag{5.4a}$$

$$\overline{newC_B} = \overline{FD_A} \otimes^* \overline{FD_B} \otimes^* (\overline{FD_A} \cap^* \overline{BD_B}) \tag{5.4b}$$

5.2.4 Selection Criteria for Cuts

So far, we have said that the *merging faces* process takes pairs of consecutive cuts for a single traversal of cuts in any main direction. However, the way and order in which such pairs are selected must be established. Four alternatives has been examined, which are explained below with 2D examples. In all of them, a traversal for the XY-sorting EVM is considered, taking cuts from lowest to highest value in the X-coordinate and supposing that there are n cuts.

Alternative 1

In this first alternative, pairs of cuts are taken in a chained form: $(C_A = C_i, C_B = C_{i+1}), i = 1, i < n, i = i + 1$, i.e., first the pair (C_1, C_2), then (C_2, C_3), and so on. In this alternative as each possible pair of cuts is considered, all cuts as well as all sections are modified. However, this alternative fails to create a valid object after applying Equation 5.4. This is due to the fact that $newC_i$ is, in general, different when it is computed from the pair (C_{i-1}, C_i) than from the pair (C_i, C_{i+1}).

Alternative 2

This case also considers pairs of cuts in a chained form, but instead of taking the i-th cut as C_A, the $newC_B$ of the last step is taken. The pair (C_1, C_2) is taken first, which generates the pair ($newC_1$, $newC_2$), then $(newC_2, C_3)$, then $(newC_3, C_4)$, and so on. Note that only in the first step this alternative is similar to the previous one, and each possible pair of cuts also is taken into account, but as C_A is the $newC_B$ of the previous step, there are no conflicts to generate a valid object. However, this alternative has a cascading effect, where some faces are dramatically extended after each step. See Figure 5.6.

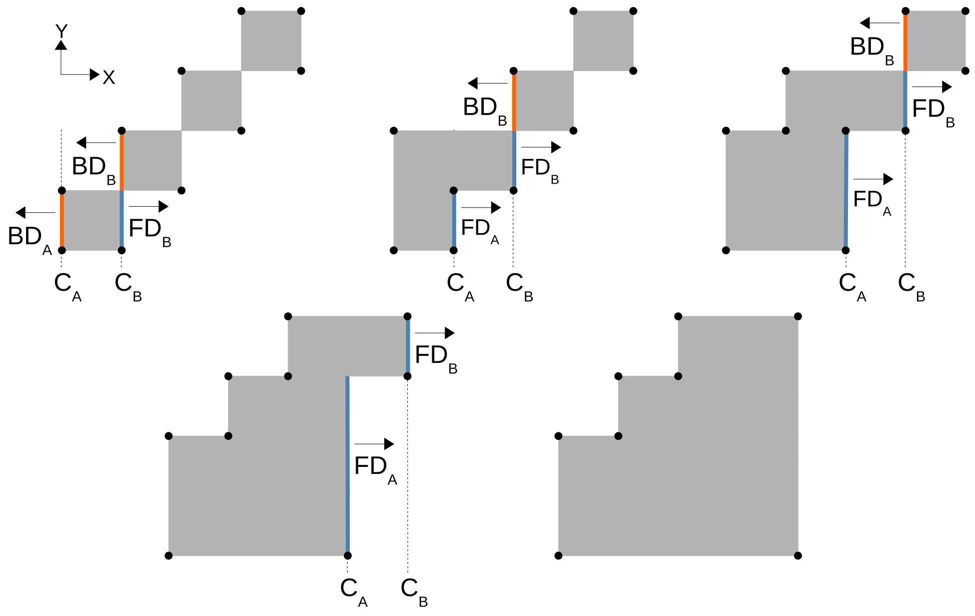


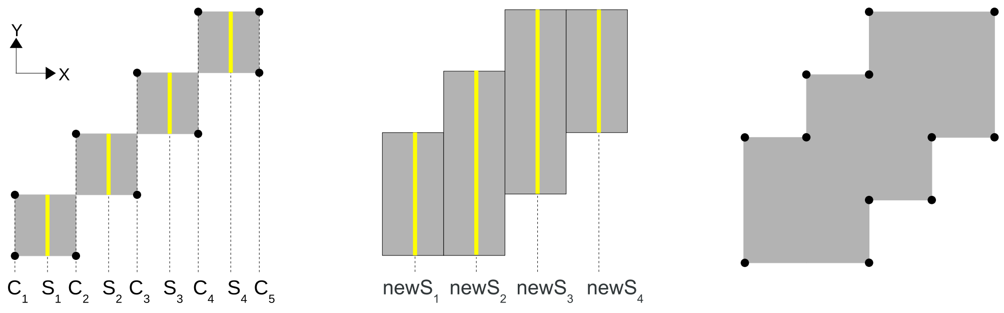
Figure 5.6: Alternative 2: Five steps for a traversal in the X-direction.

Alternative 3

This alternative consists in modifying sections instead of pairs of cuts. Pairs of cuts in a chained form are considered again, but instead of generating two new cuts, only $newC_A$ is computed using Equations 5.4 and then, a new section from $newC_A$ and the previous section of the original object using Equation 2.1 is computed, i.e.:

$$\overline{newS_i} = \overline{S_{i-1}} \otimes^* \overline{newC_A}$$

Although this alternative seems to preserve the morphology of the object, the result is similar to a morphological dilation operation, and therefore, the object expands, quickly losing its appearance. See Figure 5.7.



Y
X
C_1 S_1 C_2 S_2 C_3 S_3 C_4 S_4 C_5
newS_1 newS_2 newS_3 newS_4

Figure 5.7: Alternative 3: Sections creation for a traversal in the X-direction.

Alternative 4

The last alternative consists in taking cuts two by two $(C_A = C_i, C_B = C_{i+1}), i = 1, i < n, i = i+2$, i.e., first the pair (C_1, C_2), then (C_3, C_4), and so on. This alternative has the particularity that, although all cuts are taken into account in a single traversal, those sections between cuts that are not considered as a pair (for instance (C_2, C_3)) are not modified in the resulting object (see Figure 5.8). Theorem 3 proves this statement.

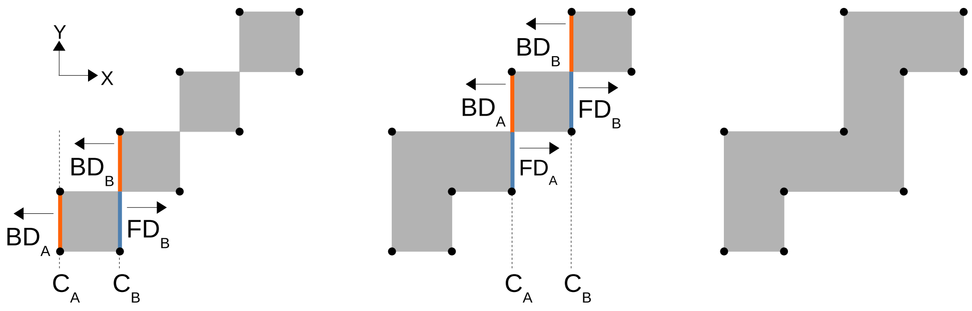


Figure 5.8: Alternative 4: Three steps for a traversal in the X-direction.

Theorem 3. *Let C_k and C_{k+1} be two consecutive cuts of an OPP P that are not considered as a pair in the merging faces process, and let S_k be the section between them. The new section, $newS_k$, between the corresponding cuts $newC_k$ and $newC_{k+1}$ is not modified, i.e., $newS_k = S_k$.*

Proof. The chosen approach to prove it is by induction.

Basis. If the first modified pair is (C_1, C_2), then, the first cuts that are not considered as pair are C_2 and C_3, so, we show that $\overline{newS_2} = \overline{S_2}$. From Equations 2.1 and 2.2, can be inferred that: $\overline{S_k(P)} = \bigotimes^{*}{}_{i=1}^{k} \overline{C_i(P)}$.

$\Rightarrow \overline{newS_2} = \overline{newC_1} \otimes^* \overline{newC_2}$

According to Equations 5.4:

$\Rightarrow \overline{newS_2} = (\overline{BD_1} \otimes^* \overline{BD_2} \otimes^* vSpace(C_1, C_2)) \otimes^* (\overline{FD_1} \otimes^* \overline{FD_2} \otimes^* vSpace(C_1, C_2))$

As $\otimes^*$ is associative and commutative operator:

$\Rightarrow \overline{newS_2} = (\overline{BD_1} \otimes^* \overline{FD_1}) \otimes^* (\overline{BD_2} \otimes^* \overline{FD_2}) \otimes^* (vSpace(C_1, C_2) \otimes^* vSpace(C_1, C_2))$

By definitions of backward and forward differences:

$\Rightarrow \overline{newS_2} = \overline{C_1} \otimes^* \overline{C_2}$

$\Rightarrow \overline{newS_2} = \overline{S_2}$

Inductive step. Let C_k and C_{k+1} be two consecutive cuts that are not considered as a pair, we assume that $newS_k = S_k$. So, the next pair that can be processed is (C_{k+1}, C_{k+2}), in this way, C_{k+2} and C_{k+3} will not be considered as a pair. Now, we show that $\overline{newS_{k+2}} = \overline{S_{k+2}}$.

$\Rightarrow \overline{newS_{k+2}} = \bigotimes^{*}{}_{i=1}^{k+2} \overline{newC_i}$

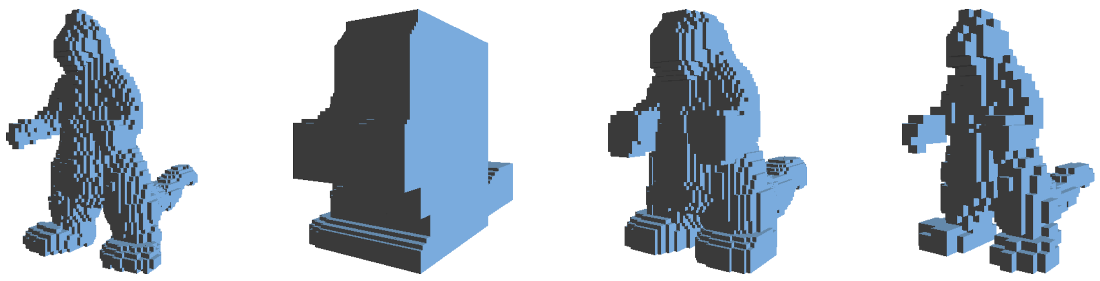
$\Rightarrow \overline{newS_{k+2}} = \overline{newS_k} \otimes^* \overline{newC_{k+1}} \otimes^* \overline{newC_{k+2}}$.

Similar to the basis step:

$\Rightarrow \overline{newS_{k+2}} = \overline{newS_k} \otimes^* (\overline{BD_{k+1}} \otimes^* \overline{BD_{k+2}} \otimes^* vSpace(C_{k+1}, C_{k+2})) \otimes^* (\overline{FD_{k+1}} \otimes^* \overline{FD_{k+2}} \otimes^* vSpace(C_{k+1}, C_{k+2}))$

$\Rightarrow \overline{newS_{k+2}} = \overline{newS_k} \otimes^* (\overline{BD_{k+1}} \otimes^* \overline{FD_{k+1}}) \otimes^* (\overline{BD_{k+2}} \otimes^* \overline{FD_{k+2}}) \otimes^* (vSpace(C_{k+1}, C_{k+2}) \otimes^* vSpace(C_{k+1}, C_{k+2}))$
$\Rightarrow \overline{newS_{k+2}} = newS_k \otimes^* (\overline{C_{k+1}} \otimes^* \overline{C_{k+2}})$
As $\overline{newS_k} = \overline{S_k}$
$\Rightarrow \overline{newS_{k+2}} = \overline{S_k} \otimes^* \overline{C_{k+1}} \otimes^* \overline{C_{k+2}}$
$\Rightarrow \overline{newS_{k+2}} = \overline{S_{k+2}}$ □

From these four alternatives, Alternative 4 was selected for the following reasons: It preserves the object's appearance better than the others. Alternative 1 is discarded for not creating a valid object. Applying alternative 2 the object is drastically deformed due to the cascading effect. Alternative 3 seems to preserve the morphology but the object expands. Figure 5.9 shows the results after running the different alternatives with one traversal for each one of three main directions in the Binzilla dataset indicating the number of EV.



(a) 4288 EV (b) 352 EV (c) 2876 EV (d) 1146 EV

Figure 5.9: Results after running the different alternatives with one traversal for each one of three main directions in the Binzilla dataset. (a) Original object. (b) Alternative 2. (c) Alternative 3. (d) Alternative 4.

5.2.5 Shape Preservation

In the presented method, merging two consecutive cuts is performed in all their extension. However, there are parts of these cuts that are isolated and merging them could result in too abrupt changes. In this section, a technique that better preserves the shape of the simplified object is devised.

Let (C_A, C_B) be the pair of cuts to be merged, we will refer as *isolated faces* those faces of C_A disjoint with C_B (on a projecting plane) and vice versa. More formally: Let $C_A = \{C_A^1, C_A^2, \dots, C_A^{n_A}\}$, $C_B = \{C_B^1, C_B^2, \dots, C_B^{n_B}\}$, $I_A = \{I_A^1, I_A^2, \dots, I_A^{n_{IA}}\}$ and $I_B = \{I_B^1, I_B^2, \dots, I_B^{n_{IB}}\}$ be the sets of faces of C_A, C_B and the isolated faces of C_A and C_B respectively. The following conditions hold for faces of sets I_A and I_B:

$$I_A \subseteq C_A, \qquad I_B \subseteq C_B \tag{5.5a}$$

$$\overline{I_A^i} \cap^* \overline{C_B} = \emptyset, \forall i = 1 \dots n_{IA} \tag{5.5b}$$

$$\overline{I_B^i} \cap^* \overline{C_A} = \emptyset, \forall i = 1 \dots n_{IB} \tag{5.5c}$$

Removing isolated faces of the merging process results in a better approximation. The corresponding new values FD'_A, BD'_A, FD'_B and BD'_B involved in the merging process are computed as:

$$\overline{FD'_A} = \overline{FD_A} -^* \overline{I_A}, \qquad \overline{BD'_A} = \overline{BD_A} -^* \overline{I_A} \tag{5.6a}$$

$$\overline{FD'_B} = \overline{FD_B} -^* \overline{I_B}, \qquad \overline{BD'_B} = \overline{BD_B} -^* \overline{I_B} \tag{5.6b}$$

these values are used in Equations 5.4 to generate $newC'_A$ and $newC'_B$, such that:

$$\overline{newC'_A} = \overline{BD'_A} \otimes^* \overline{BD'_B} \otimes^* (\overline{FD'_A} \cap^* \overline{BD'_B}) \tag{5.7a}$$

$$\overline{newC'_B} = \overline{FD'_A} \otimes^* \overline{FD'_B} \otimes^* (\overline{FD'_A} \cap^* \overline{BD'_B}) \tag{5.7b}$$

after that, the isolated faces must be reintegrated. This process can be accomplished with a union operation, however, by definition I_A and C_A are disjoint sets and therefore, I_A and $newC'_A$ are also disjoint sets (the same applies to I_B, C_B and $newC'_B$). Then, according to the EVM Property 5, an XOR operation is performed instead.

$$\overline{newC_A} = \overline{newC'_A} \otimes^* \overline{I_A} \tag{5.8a}$$

$$\overline{newC_B} = \overline{newC'_B} \otimes^* \overline{I_B} \tag{5.8b}$$

Figure 5.10 shows an example where an isolated face (*if*) is depicted. Here, *if* remains in place, and only those faces that share either an edge or a vertex are taken into account throughout the *merging faces* process. The application of this technique does not affect to the aforementioned *subset property* (see Section 5.2.1) and produces better approximations, for instance see Figure 5.11.

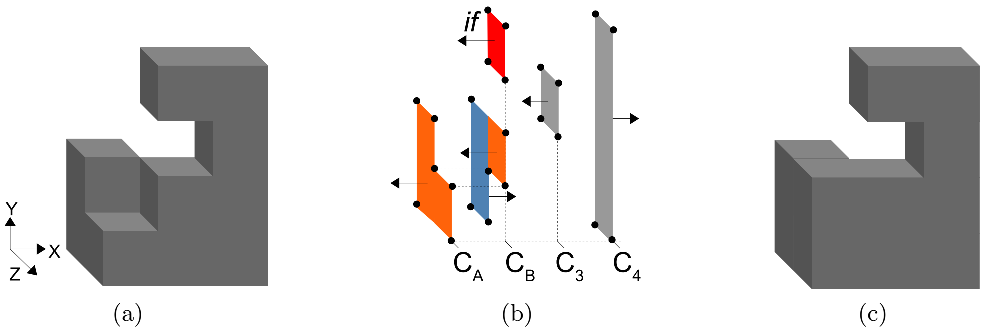
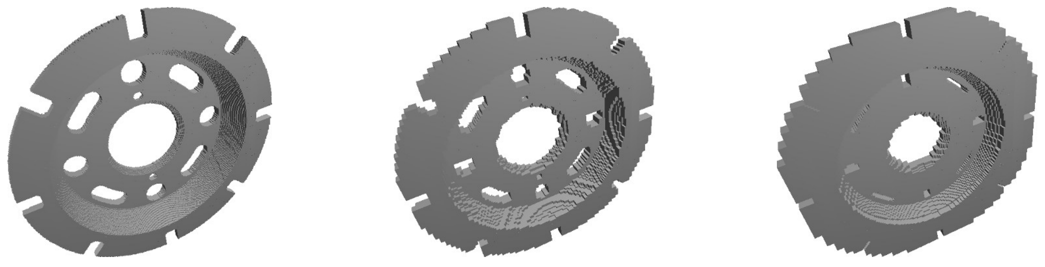


Figure 5.10: Shape preservation example . (a) A simple 3D OPP. (b) All cuts and extreme vertices; in red, an isolated face. (c) Result of applying *merging faces* with the technique to the pair (C_A, C_B).



(a) 55246 EV (b) 6512 EV (c) 7320 EV

Figure 5.11: Shape preservation technique in the DiskBrake dataset. (a) Original dataset. (b) and (c), resulting BOPP using and not using the technique respectively. Both BOPP have less than 15% of EV of the original dataset.

Isolated Faces Detection

Detecting isolated faces in 2D is straightforward by displacing the brinks in both cuts and checking for intersection. In the 3D case, the sets of elements to analyze are 2D faces, i.e., general orthogonal polygons. So, a different strategy is applied to individually separate each face of C_A and C_B to detect the isolated ones.

Cuts C_A and C_B are converted to CUDB and their faces are extracted, and to check for intersection, the CUDB collision detection is used (see Section 3.2.6). Algorithm 14 details the steps for the isolated faces detection process, where function *extractFaces*() extracts each connected component as an individual CUDB model, and in the set of faces F, a face f_i, is a 2D-CUDB with an additional attribute that indicates the cut to which the face belongs.

Algorithm 14: getIsolatedFaces

```
Input  : $C_A, C_B$;                                   /* Cuts (2D-EVM) /*
Output: $I_A, I_B$;                  /* Set of isolated faces of $C_A$ and $C_B$ /*
$obj_A \leftarrow EVMtoCUDB(C_A)$;  $obj_B \leftarrow EVMtoCUDB(C_B)$;
$CCL(obj_A)$;  $CCL(obj_B)$;
$F \leftarrow extractFaces(Ojb_A) \cup extractFaces(Ojb_B)$;
$S \leftarrow detectCollision(F)$;                       /* Algorithm 6 /*
$F_A \leftarrow$ Faces $f_i \in F$ belonging to $Obj_A$ such that $f_i \notin S$;
$F_B \leftarrow$ Faces $f_i \in F$ belonging to $Obj_B$ such that $f_i \notin S$;
$I_A \leftarrow CUDBtoEVM(F_A)$;  $I_B \leftarrow CUDBtoEVM(F_B)$;
```

5.2.6 Merging Faces Algorithm

According to Alternative 4 for cuts selection, the *merging faces* strategy takes cuts two by two, first the pair (C_1, C_2), then (C_3, C_4), and so on. But, observe the particular case in Figure 5.5(b), where if $C_A = C_1$ and $C_B = C_2$, $newC_A$ and $newC_B$ will be exactly the same as C_A and C_B respectively. This is due to the fact that $FD_A = \emptyset$ and $BD_B = \emptyset$, which is the condition that results in any displacement. The direct application of this condition to Equation 5.4 shows this conclusion.

On the other hand, when the shape preservation technique is applied, there may be consecutive cuts with all its faces detected as isolated (see Figure 5.12), in such cases, there will be no displacement. Note that, the condition $I_A = C_A$ is enough to determine that all faces are isolated as it implicitly means that $I_B = C_B$, and vice versa.

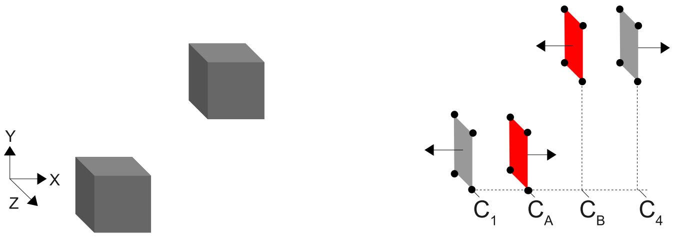


Figure 5.12: Example where all elements of C_A and C_B are isolated.

Moreover, the merging process must be controlled by a maximum displacement allowed, in such a way that only pairs of consecutive cuts that are at a distance less or equal than a displacement parameter, d, are actually merged. Without this condition, far apart consecutive cuts can be merged causing abrupt changes. Therefore, given the pair of cuts $(C_A = C_i, C_B = C_{i+1})$ the conditions to merge them are:

$$(FD_A \neq \emptyset \text{ or } BD_B \neq \emptyset) \tag{5.9a}$$

$$\text{Not all faces of } C_A \text{ and } C_B \text{ are isolated} \tag{5.9b}$$

$$distance(C_A, C_B) \leq d \tag{5.9c}$$

If conditions in Expression 5.9 are fulfilled, the pair (C_i, C_{i+1}) is processed with Equations 5.8 to compute $newC_A$ and $newC_B$, and the method continues, according to the selection criteria, with the pair (C_{i+2}, C_{i+3}). Otherwise C_i is copied directly to the simplified object and the method continues with the pair (C_{i+1}, C_{i+2}).

Nevertheless, there may be cases where all faces of all the possible pairs of cuts in the object are isolated (see Figure 5.12), which means that the method has no effect and the complete simplification process will never end. Therefore, in order to guarantee the property of finiteness, when all faces of a given pair of cuts are isolated, the shape preservation technique is not applied and Equations 5.4 instead of Equations 5.8 are used for such pair.

Another issue that has been considered is that, if cuts are analyzed in descending order along one main direction, the result is barely different with respect to the result in ascending order. Therefore, for symmetry preservation purposes the object is analyzed from both sides at a time. However, this symmetrical traversal does neither improve nor reduce the performance of the method.

Algorithm 15 shows the steps of the *merging faces* using a lazy evaluation strategy to improve its performance. A given OPP P is ABC-sorted according to a given *sorting* and then it is processed in the A-direction with a given parameter of distance d, and a flag to indicate if cuts with all its faces as isolated are displaced or not.

Algorithm 15: mergingFaces

```
Input  : $P$;                                          /* EVM-represented OPP /*
Input  : $sorting$;                                    /* ABC-sorting /*
Input  : $d$;                                          /* Displacement parameter /*
Input  : $displaceIF$;          /* Flag to allow displacing cuts having all its faces isolated /*
Output: $Q$;                                           /* EVM-represented simplified OPP /*
$Q \leftarrow \emptyset$;  $Q.sorting \leftarrow P.sorting$;
$C_A, coord_A \leftarrow P.getNextCut()$;  $C_B, coord_B \leftarrow P.getNextCut()$;
$C_D, coord_D \leftarrow P.getReverseNextCut()$;  $C_C, coord_C \leftarrow P.getReverseNextCut()$;
$ascendingOrder \leftarrow$ true;
while $coord_B < coord_C$ do
    $process \leftarrow$ false;
    if $ascendingOrder$ then
        if $distance(C_A, C_B) <= d$ then              /* Merging condition 5.9c /*
            Compute $FD_A$ and $BD_B$;
            if $FD_A \neq \emptyset$ or $BD_B \neq \emptyset$ then    /* Merging condition 5.9a /*
                $I_A, I_B \leftarrow getIsolatedFaces(C_A, C_B)$;
                if $I_A \neq C_A$ then                    /* Merging condition 5.9b /*
                    $FD_A \leftarrow FD_A -^* I_A$,  $BD_B \leftarrow BD_B -^* I_B$;
                    if $FD_A \neq \emptyset$ or $BD_B \neq \emptyset$ then  /* Merging condition 5.9a /*
                        Compute $FD_B$ and $BD_A$;
                        $FD_B \leftarrow FD_B -^* I_B$,  $BD_A \leftarrow BD_A -^* I_A$;
                        $process \leftarrow$ true;
                    end if
                else                                      /* All faces are isolated /*
                    if $displaceIF$ then
                        Compute $FD_B$ and $BD_A$;  $process \leftarrow$ true;
                    end if
                end if
            end if
        end if
        if $process$ then                                 /* Formula application /*
            $voidSpace \leftarrow FD_A \cap^* BD_B$;
            $newC_A \leftarrow BD_A \otimes^* BD_B \otimes^* voidSpace$;
            $newC_B \leftarrow FD_A \otimes^* FD_B \otimes^* voidSpace$;
            if $I_A \neq C_A$ then                        /* Not all faces are isolated /*
                $newC_A \leftarrow newC_A \otimes^* I_A$;  $newC_B \leftarrow newC_B \otimes^* I_B$;
            end if
            $Q.insertCut(newC_A)$;  $Q.insertCut(newC_B)$;
            $C_A, coord_A \leftarrow P.getNextCut()$;  $C_B, coord_B \leftarrow P.getNextCut()$;
        else                                              /* Omit and copy cut /*
            $Q.insertCut(C_A)$;
            $C_A \leftarrow C_B$;  $C_B, coord_B \leftarrow P.GetNextCut()$;
        end if
    else
        ***Similar operations for descending order with $C_C$ and $C_D$
    end if
    $ascendingOrder \leftarrow \neg ascendingOrder$;
end while
```

5.3 Lossy Simplification

5.3.1 Initial Considerations

According to the *merging faces* algorithm, when applying the shape preservation technique, if there are pairs of cuts with all its faces isolated, there are two options, one of them preserves the shape and the other one guarantees the finiteness of the method. Therefore, in order to take advantage of the shape preservation technique and still guarantee that the method ends, the strategy is to perform first three times the *merging faces* process, once per each main direction (having previously determined the best sorting) with a given parameter of distance d, and not allowing to displace isolated faces when all faces are detected as such. Then, three additional *merging faces* process with the same parameter d are performed, but now allowing to displace the isolated faces.

Besides, after applying the *merging faces* process in the three main directions with a given distance d, cuts with the same distance d between them may still remain. This is because some faces of the intermediate cuts that are not considered as pairs (see Section 5.2.4) or that were marked as isolated and not displaced, were not modified after the merging in the three main directions. So, with the aforementioned three additional traversals, the approximations have a good quality and the finiteness property is guaranteed.

5.3.2 Simplification Algorithm

Let Φ_0 be an initial OPP. A finite sequence $\Phi_1, \Phi_2, \ldots, \Phi_p$ of OPP can be generated, that fulfills the following properties:

1. $\Phi_{i+1} = B(\Phi_i)$, $\forall i = 0 \ldots p-1$
2. $\Phi_i \subseteq B(\Phi_i)$, $\forall i = 0 \ldots p-1$, and, therefore, $\Phi_i \subseteq \Phi_{i+1}$, $\forall i = 0 \ldots p-1$
3. $\Phi_p = AABB(\Phi_i)$, $\forall i = 0 \ldots p$

The first property indicates that the approach is incremental. The second one, corresponds to the *subset property* for bounding structures. Since the face displacements are done outward of the object, and are restricted by the cuts of the initial object, the resulting OPP will never extend beyond these cuts and, therefore, it will contain the initial object.

The last property states that the sequence is finite and that ends with the AABB that is shared by all of the OPP in the sequence. The AABB property can be demonstrated by considering that the AABB of an OPP can be defined as the intersection of the six planes corresponding to the first and last cut in the three main directions and the fact that the object will never extend beyond the initial cuts, and, in particular, the first and last ones. Moreover, it can be observed that the first cut only has BD faces and the last one only FD faces. Then, in each iteration the first cut of $B(\Phi_i)$ will be either the same of Φ_i or the union of the first cut and the BD of the second cut of Φ_i. Then, in the last iteration, the first cut of Φ_p, i.e., of the AABB, corresponds to the union of all of the BD of all cuts. The same reasoning applies for the last cut.

According to these properties, the generated sequence seems to have similarity with a hierarchy of orthogonal visual hulls [53]. The visual hull is a concept of 3D reconstruction by shape-from-silhouette technique, where a 3D representation of an object is created by its silhouettes within several images from different viewpoints. Each silhouette forms in their projection a cone. The visual hull of an object is the envelope of all its possible circumscribed cones [81]. As the number of cones increases, the object is reconstructed with higher precision.

As the proposed method is incremental, it actually computes all the objects between Φ_1 and Φ_k for successive values of d, $d = d_i, i = 1, ..., k$, obtaining the corresponding Φ_i. Displacements d_i can be in any units and incremented in any quantity. For OPP corresponding to digital images the basic unit is one voxel, i.e. $d_i = i$. For general OPP, with float coordinate values, d_i can take any values ranging from the minimum distance between cuts and the AABB size.

The input of the method is an EVM-represented OPP, P, which corresponds to the initial object Φ_0, and we have considered two queries of simplification:

- Give the maximum value for the displacement, d_{max}, i.e. the method performs iteratively the *merging faces* process and stops when $d > d_{max}$.
- Give the maximum number of EV nv in the last BOPP, i.e. the sequence ends with the object Φ_k, which has no more than nv extreme vertices. In the particular case of ending with the AABB, i.e. $o_k = o_p = AABB(P)$, nv must be 8.

Steps for the two queries in the proposed simplification method are shown in Algorithm 16 (maximum value for the displacement) and Algorithm 17 (maximum number of EV), where function *detectBestSorting*() detects the best sorting (see Section 5.2.1). For simplify purposes, in both cases we suppose an increment of 1 voxel. Note that Algorithm 16 performs 6 times the *merging faces* process per each distance. But, in Algorithm 17 the method can stop even if the last three *merging faces* processes for the same distance are not performed. This intermediate object is an OPP that has been simplified in the three main directions.

Algorithm 16: EVMSimplification1

```
Input  : $P$;                                          /* EVM-represented OPP /*
Input  : $d_{max}$;                      /* Maximum value for the displacement /*
Output: $Q$;                               /* EVM-represented simplified OPP /*
$Q \leftarrow P$;
$sort1, sort2, sort3 \leftarrow detectBestSorting()$;
for $d \leftarrow 1$ to $d_{max}$ do
    $Q \leftarrow mergingFaces(Q, sort1, d,$ false$)$;
    $Q \leftarrow mergingFaces(Q, sort2, d,$ false$)$;
    $Q \leftarrow mergingFaces(Q, sort3, d,$ false$)$;
    $Q \leftarrow mergingFaces(Q, sort1, d,$ true$)$;
    $Q \leftarrow mergingFaces(Q, sort2, d,$ true$)$;
    $Q \leftarrow mergingFaces(Q, sort3, d,$ true$)$;
    if $Q.getNEV() = 8$ then break; end if;
end for
```

Algorithm 17: EVMSimplification2

```
Input  : $P$;                                          /* EVM-represented OPP /*
Input  : $nv$;                             /* Maximum number of EV in result /*
Output: $Q$;                               /* EVM-represented simplified OPP /*
$Q \leftarrow P$;  $displaceIF \leftarrow$false;  $d \leftarrow 1$;
$sort1, sort2, sort3 \leftarrow detectBestSorting()$;
while $Q.get\_nev() > nv$ do
    $Q \leftarrow mergingFaces(Q, sort1, d, displaceIF)$;
    $Q \leftarrow mergingFaces(Q, sort2, d, displaceIF)$;
    $Q \leftarrow mergingFaces(Q, sort3, d, displaceIF)$;
    if $displaceIF$ then $d \leftarrow d + 1$; end if;
    $displaceIF \leftarrow \neg displaceIF$;
end while
```

5.4 Quality of the Approximations

The presented approach generates bounding volumes, where every object Φ_{i+1} bounds object Φ_i as tightly as possible. The quality (tightness) $\Theta(\Phi_i)$ of a BOPP Φ_i is measured as the volume difference between Φ_i and the original OPP Φ_0, divided by the volume difference between the volume of the AABB (Φ_p) and Φ_0, i.e.:

$$\Theta(\Phi_i) = \frac{V(\Phi_i) - V(\Phi_0)}{V(\Phi_p) - V(\Phi_0)}, \quad \forall i = 0 \ldots p \tag{5.10}$$

where $V(\Phi_i)$ is the volume of Φ_i.

This function is useful for indicating the distortion of the resulting BOPP, where $\Theta(\Phi_i)$ ranges from 0 (original Φ_0) to 1 (AABB Φ_p).

5.5 Lossless Progressive Encoding

In this section, the Progressive Extreme Vertices Encoding (PEVE) is described. It is a data structure that encodes a BOPP sequence in a progressive and lossless way, with a reduced storage size.

5.5.1 EVM Encoding

In its raw format, EVM represents vertices as 3D points. Then, a 3D OPP is represented with a set of $3n$ values, n being the number of extreme vertices (EV). In storage size, this set occupies $3nb$ bits, b being the number of bits to store a vertex coordinate.

EVM allows different schemes for data compression using the fact that EV are arranged in an ordered set of cuts [1]. There is not a general scheme since for each object it depends on its shape and the ABC-ordering. Considering an ABC-ordering, the common A-coordinates, for EV lying on each A-cut, and the common B-coordinates, for EV lying on each AB-cut, can be factored out. This compressed EVM uses 2 flags (2 bits) per each vertex to indicate respectively if the A and B-coordinates are factored out and thus, these values are not saved. Therefore, a vertex requires at most $3b+2$ bits when no coordinate is factored out, and only $b+2$ bits when both the A and B-coordinates are factored out.

5.5.2 Progressive Extreme Vertices Encoding (PEVE)

Throughout the *merging faces* process, there can be EV of object Φ_i that remain in object Φ_{i+1}. Let C_A and C_B be a pair of consecutive cuts, the following sets of vertices are remaining vertices:

- Vertices belonging to $(EVM(\overline{BD_A}) - EVM(\overline{BD_B}))$ or to $(EVM(\overline{FD_B}) - EVM(\overline{FD_A}))$. This fact can be proved by applying directly Equations 5.4.
- Vertices of the isolated faces of BD_A and FD_B when the shape preservation technique is applied.

Therefore, PEVE encodes the sequence of objects in such a way that remaining vertices are stored only once. For instance, Figure 5.13 depicts a 2D LOD sequence generated by the presented approach. For this dataset, the method produces 3 more objects. Note that there are some remaining EV. Figure 5.14 depicts the sequence put together and each EV is labeled with the object numbers to which it belongs. Note that, although there are vertices that only appear once, there are also vertices that appear in two, three and even in the four objects. Moreover, most of these vertices appear in consecutive objects, and only one appears in non-consecutive objects. For instance, the vertex in the upper-left corner of the object appears in all the objects, the vertex in the bottom-right corner appears from object 1 to 3, while the vertex highlighted in yellow appears only in objects 0 and 2.

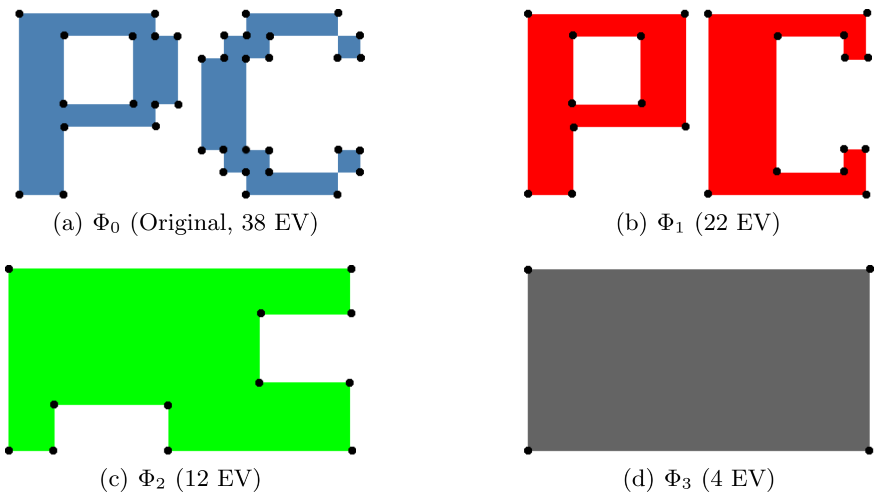

(a) Φ_0 (Original, 38 EV)
(b) Φ_1 (22 EV)
(c) Φ_2 (12 EV)
(d) Φ_3 (4 EV)

Figure 5.13: Objects generated by the simplification method in a 2D dataset, indicating on each one the number of extreme vertices (EV), which are marked with black dots. (a) Original Φ_0, (b) Φ_1, (c) Φ_2, (d) Φ_3=AABB.

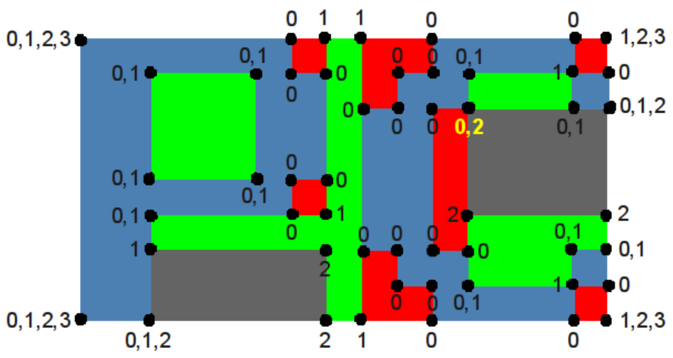


Figure 5.14: Sequence of the Figure 5.13 put together. Each extreme vertex is labeled with the numbers of the objects which it belongs

PEVE encodes EV of all the generated objects into a single set, by storing, for any vertex coordinate, an indicator to which objects this EV belongs. For storage purposes, EV are classified into the next categories:

Unique (U): it appears once, i.e., it belongs only to one object Φ_i of the sequence.

Consecutive (C): it has a consecutive occurrence, i.e., it belongs to all the objects in an object interval $[\Phi_i, \Phi_j]$, $j > i$.

Sporadic (S): it appears several times, either as an U or a C-vertex.

In a sporadic EV, the number of times it can appear is no more than two in 2D and four in 3D as the following theorem proves.

Theorem 4. *Let v be an extreme vertex in the LOD sequence, the number of non successive occurrences throughout the simplification process is no more than two in 2D, and four in 3D.*

Proof. The proof is performed by enumeration. According to the clasification of OPP vertices (see Section 2.2.4), a vertex v is an EV if it has an odd number of black and white surrounding voxels (see Figure 2.5). Taking into account that *merging faces* always displaces faces outward the object, i.e. it enlarges the object, it can gradually go from 0 to 8 surrounding voxels, and not vice versa. Therefore, in the worst case scenario, if the number of surrounding voxels of v changes step by step, v can be EV at most four times (in odd configurations). The 2D case can be proved with the same reasoning. $\square$

Furthermore, experimental tests have yielded that sporadic vertices appear rather infrequently, less than 6% of all the EV in the LOD sequence (see Table 5.2).

In order to encode the aforementioned information, the next *v-scheme* is used in PEVE. For every vertex 2 bits are used to indicate its type, plus the following associated information that depends on the number of objects of the sequence, p:

- U-vertex: the object number to which it belongs, i.e. $log_2(p)$ bits.
- C-vertex: the number of the first and last objects to which it belongs, i.e. $2log_2(p)$ bits.

- S-vertex: 2 bits to indicate the number of intervals, plus 1 bit for each interval to indicate its type, followed by the bits for the interval information. In the worst case scenario S-vertex can appear 4 times as C-vertex, i.e., it requires at most $2 + 4(1 + 2log_2(p)) = 6 + 8log_2(p)$ bits.

5.5.3 Data Structure for Progressive Encoding

In order to progressively decode or transmit the whole LOD sequence, PEVE must store first those EV that belong to object Φ_p, then to Φ_{p-1}, and so on. Therefore, EV are sorted in the following way:

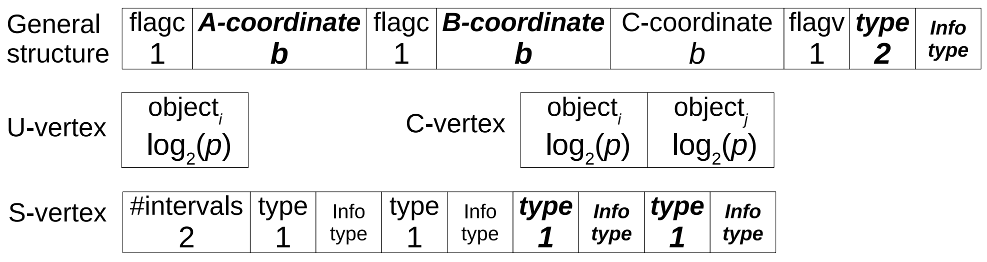
1. First, according to the last object they belong in descending order.
2. Then, by type (first S, then C, and then U-vertices).
3. Finally, by its X,Y, and Z-coordinates in ascending order.

In this way, vertices corresponding to the more simplified object can be decoded first, then objects with a better LOD are progressively decoded, yielding at the end the lossless information of the original object Φ_0.

Vertices are grouped by type since several U and C-vertices share the same *v-scheme* with other vertices of the same type. Then, using a flag (1 bit) to indicate if the *v-scheme* is factored out, this information can be omited. The number of bits needed for a vertex coordinate is obtained from the resolution size of the dataset. Figure 5.15 depicts this data structure.

Figure 5.16 shows PEVE in ASCII format for the dataset shown in Figure 5.13. The whole sequence has 76 vertices but only 51 vertices are encoded. After each vertex, letters U, C and S indicate the type, followed by its *v-scheme*; symbol * indicates that *v-scheme* or a coordinate value is factored out. Moreover, note that, when the first vertex whose last object is Φ_0 appears (vertex 29), it is not necessary to save the *v-scheme* of the following vertices, as all of them are U-vertices belonging to Φ_0, and therefore, this information can be omitted in the encoding.

Following with this example and considering 4 bits to code a vertex coordinate, vertex 5 is coded with $1+4+4+1+2+2+1+2+1+2 = 20$ bits, vertex 13 with $1+4+4+1+2+2+2 = 16$



General structure	flagc	***A-coordinate***	flagc	***B-coordinate***	C-coordinate	flagv	***type***	***Info type***
	1	***b***	1	***b***	*b*	1	***2***	

U-vertex	object$_i$
	$log_2(p)$

C-vertex	object$_i$	object$_j$
	$log_2(p)$	$log_2(p)$

S-vertex	#intervals	type	Info type	type	Info type	***type***	***Info type***	***type***	***Info type***
	2	1		1		***1***		***1***	

Figure 5.15: Data structure for PEVE, indicating the number of bits required for each element. Top: The general structure, *flagc* indicates if the coordinate value is factored out, *flagv* indicates if the *v-scheme* is factored out, *b* is the number of bits to encode a vertex coordinate. Bottom: structure for the three types of vertices, where *p* is the number of encoded objects. Data that can eventually be excluded is highlighted in bold and italic font.

```
 1:  (16,1) C1-3        14:  ( *,5) *       27:  (15,2) *       40:  ( *,8)
 2:  ( *,9) *           15:  ( *,8) *       28:  ( *,8) *       41:  (11,1)
 3:  ( 1,1) C0-3        16:  ( 6,5) *       29:  ( 7,4) U0      42:  ( *,2)
 4:  ( *,9) *           17:  ( *,8) *       30:  ( *,5)         43:  ( *,3)
 5:  (12,7) S2 U0 U2    18:  (12,2) *       31:  ( *,8)         44:  ( *,7)
 6:  ( 3,1) C0-2        19:  ( *,8) *       32:  ( *,9)         45:  ( *,8)
 7:  (16,7) *           20:  (15,3) *       33:  ( 8,5)         46:  ( *,9)
 8:  ( 3,3) U2          21:  ( *,7) *       34:  ( *,8)         47:  (12,3)
 9:  ( 8,1) *           22:  (16,3) *       35:  ( 9,3)         48:  (15,1)
10:  ( *,3) *           23:  ( 8,4) U1      36:  ( *,7)         49:  ( *,9)
11:  (12,4) *           24:  ( *,9) *       37:  (10,2)         50:  (16,2)
12:  (16,4) *           25:  ( 9,1) *       38:  ( *,3)         51:  ( *,8)
13:  ( 3,4) C0-1        26:  ( *,9) *       39:  ( *,7)
```

Figure 5.16: ASCII representation of PEVE for the sequence of objects in Figure 5.14

bits, vertex 14 with $1 + 4 + 1 = 6$, and vertex 30 with $1 + 4 = 5$ bits. The whole LOD sequence requires 54 bytes. Encoding each object separately with the compressed EVM (see Section 5.5.1) requires 61 bytes, and with EVM in its raw form ($2 \cdot 4$ bits per vertex), 76 bytes. Therefore, in the first case the compression ratio is 0.88 and in the second case 0.71.

5.6 Experimental Results

The simplification method with PEVE has been tested on several 3D datasets with different shape features such as size, number of connected components, concavities and holes. All datasets come from public volume repositories or own collection. The corresponding programs have been written in C++ and tested on a PC Intel®Core 2 Duo CPU E6600@2.40GHz with 3.2 GB RAM and running Linux.

The results of simplification reported here have been computed using Algorithm 17 with $nv = 8$ and storing all the intermediate objects with PEVE. Statistics for a compilation of twelve datasets are listed in Table 5.1. All these datasets are shown in Figure 5.11, and Figures 5.18 to 5.28. Results of the encoding of the LOD sequence are listed in Table 5.2. For these datasets the number of bits to encode a vertex coordinate varies from 7 to 11. Note that the number of encoded objects is not $2d_{max}$ because there can be repeated generated objects, and they are not consider as a new object.

Figure 5.17(a) depicts the quality distortion curves of the test datasets as a function of the percentage of total bytes decoded (transmitted) in order to get the original object. This graph shows that the presented approach yields good approximations, $\Theta < 0.2$, with only 10% of the decoded data. Figure 5.17(b) gives a closeup of the first 10%. Note that the more orthogonal or bigger is the object the less distortion it has along the simplification process (e.g., Pelvis, PhlegmaticDragon and DoorPieces), while the opposite happens with sparse or small objects (e.g., Aneurysm, Venus and DiskBrake).

Table 5.1: Statistics of the test datasets. For each dataset: size in voxels, number of extreme vertices ($|EV|$) in the original dataset, maximum used distance (d_{max}), number of processed pairs ($|pairs|$), i.e., number of times Equations 5.4 is executed, and global computation time to simplify the object until the AABB (in seconds).

Dataset	Size	$\|EV\|$	d_{max}	$\|pairs\|$	Time (s.)
DoorPieces	268x72x394	11784	42	1001	0.87
Venus	55x55x192	15636	16	834	0.92
FanDisk	275x300x153	21010	64	1405	1.88
Bunny	127x128x98	24796	34	1219	1.86
Aneurysm	213x215x240	50318	40	6013	13.61
Foot	110x310x112	51,532	45	2613	4.79
Diskbrake	299x300x43	55246	32	2410	4.51
Engine	139x197x108	101114	17	2990	10.16
Ramesses	178x512x268	117840	38	4493	10.41
Athene	350x195x512	179400	73	4052	13.10
Phleg.Dragon	781x436x603	716840	124	16143	110.80
Pelvis	905x1259x1108	2082078	206	42665	471.30

Table 5.2: Encoding results of the test datasets. For each dataset: total number of generated objects ($|O|$), total number of extreme vertices ($|TEV|$), total PEVE storage size in bytes, compression rates with respect to a direct storage of the sequence in its raw EVM (cr_r) and with respect to the compressed EVM (cr_c) (see Section 5.5.1), and percentage of sporadic vertices (%S-v).

Dataset	$\|O\|$	$\|TEV\|$	bytes	cr_r	cr_c	%S-v
DoorPieces	37	19948	41152	0.44	0.85	2.6
Venus	22	24803	43223	0.43	0.88	3.6
FanDisk	40	35166	78034	0.51	0.90	3.0
Bunny	29	41335	71153	0.47	0.85	3.6
Aneurysm	52	152691	315686	0.40	0.73	3.2
Foot	40	92834	198925	0.44	0.81	4.1
DiskBrake	34	108101	228187	0.45	0.84	4.4
Engine	29	171562	311643	0.41	0.79	4.3
Ramesses	41	204039	424275	0.43	0.81	3.6
Athene	54	286099	573122	0.39	0.77	4.3
Phleg.Dragon	81	1382414	3236326	0.36	0.69	5.3
Pelvis	119	4762211	11672096	0.31	0.61	5.3

5.7 Comparison with Other Methods

An ideal comparison would be against other methods with the same dataset at the same scale, computing compression rates and evaluating the visual quality. However, as was pointed out in Section 2.4, there are few methods that produce a progressive LOD sequence of bounding volumes, and their corresponding results are reported in different ways making a global comparison hard. Nonetheless, those three methods in the revised literature that are more related to presented one have been selected to perform ad-hoc comparisons (see Section 2.4.1).

The presented approach can generate sequences of objects in which pairs of consecutive objects only change in very few vertices. However, we are aware that for progressive methods,

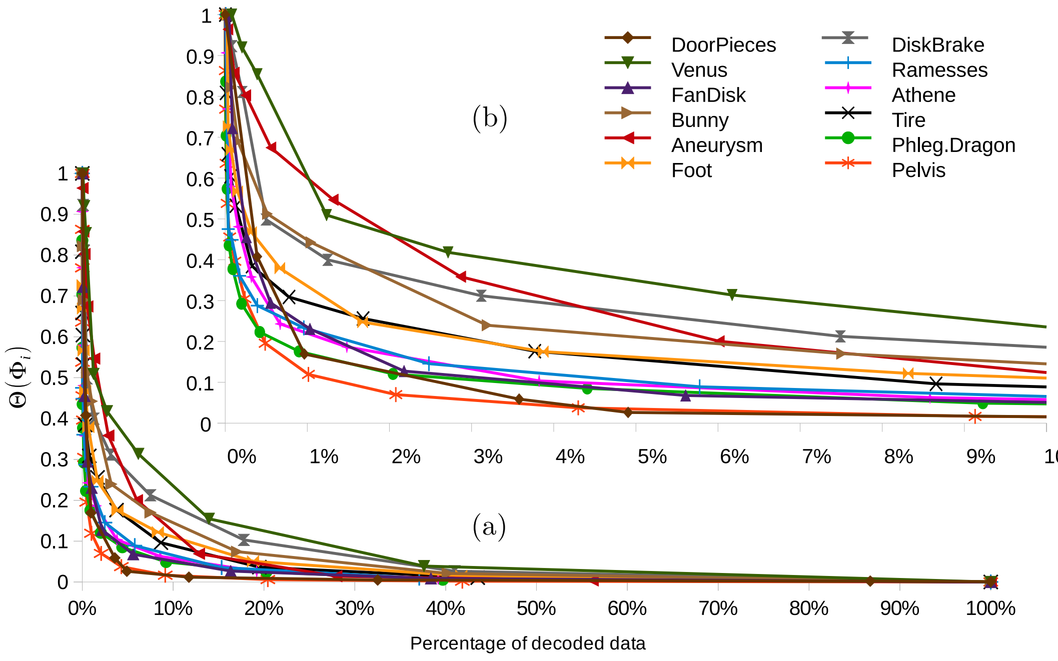


Figure 5.17: Quality distortion curves as a function of the percentage of total bytes decoded.

logarithmically spaced approximations are quite good. Therefore, in the following comparisons this kind of sequences have been generated, i.e., consecutive objects that satisfy the following statement: $|EV(\Phi_i)| \leq 0.5|EV(\Phi_{i-1})|, \forall i = 1 \ldots p$. Table 5.3 shows the statistics for the test datasets with this restriction.

Table 5.3: Statistics of the test datasets storing logarithmically spaced approximations. For each dataset: total number of generated objects ($|O|$), total number of extreme vertices ($|TEV|$), total PEVE storage size in bytes, compression rates with respect to a direct storage of the sequence in its raw EVM (cr_r) and with respect to the compressed EVM (cr_c).

Dataset	$\lvert O\rvert$	$\lvert TEV\rvert$	bytes	cr_r	cr_c
DoorPieces	9	16019	32895	0.55	0.98
Venus	9	20388	31477	0.46	0.98
FanDisk	10	28535	58865	0.56	0.99
Bunny	10	33298	51698	0.53	0.98
Aneurysm	12	81461	140685	0.49	0.96
Foot	12	70658	135211	0.51	0.98
DiskBrake	10	81462	140451	0.51	0.98
Ramesses	11	154473	291935	0.51	0.98
Athene	13	239607	432231	0.47	0.97
Tire	13	400328	695240	0.46	0.98
PhlegmaticDragon	15	968955	1972880	0.48	0.97
Pelvis	16	2871608	6170999	0.46	0.97

5.7.1 Comparison with an Octree-based Method

The Bunny dataset has been tested by Samet and Kochut in an octree-based approximation method [135]. The size of the volume data is 128^3 and, therefore, the region octree has a maximum depth of 7 and the method generates a LOD sequence of 8 objects. In this method, 29007 blocks are necessary to reconstruct the original Bunny. Each block is uniquely identified with a *locational code*, z_n, which for any node x_m is an integer computed with the next recursive expression:

$$z_i = \begin{cases} 0 & i = m \\ 9 \cdot z_{i-1} + \mathit{childtype}(x_i) & m < i \leq n \end{cases}$$

where the sequence $< x_i >$ represents all the directional codes on the path from the root node x_n to node x_m and *childtype* returns the corresponding directional code.

Although this locational code supposes some compression degree, the authors do not explain any way for further compression. However, the generated size of both codifications has been compared in two different ways. First, for an octree with depth 7, in the worst case scenario when all the directional codes are 8, this expression gives a maximum value of $8 \cdot 9^0 + 8 \cdot 9^1 + \cdots + 8 \cdot 9^6 = 4782968$, therefore, the locational code for each block will require at most 23 bits. Thus, we estimate that the sequence of blocks of the Bunny requires, in this case, a total storage size of 83396 bytes (29007 blocks $\cdot$ 23 bits). For the same dataset and at the same resolution, PEVE requires 51698 bytes for a total of 33298 EV that encode 10 objects. Note that in this case a vertex coordinate is encoded with $log_2(128) = 7$ bits. This storage size is 0.619 of the

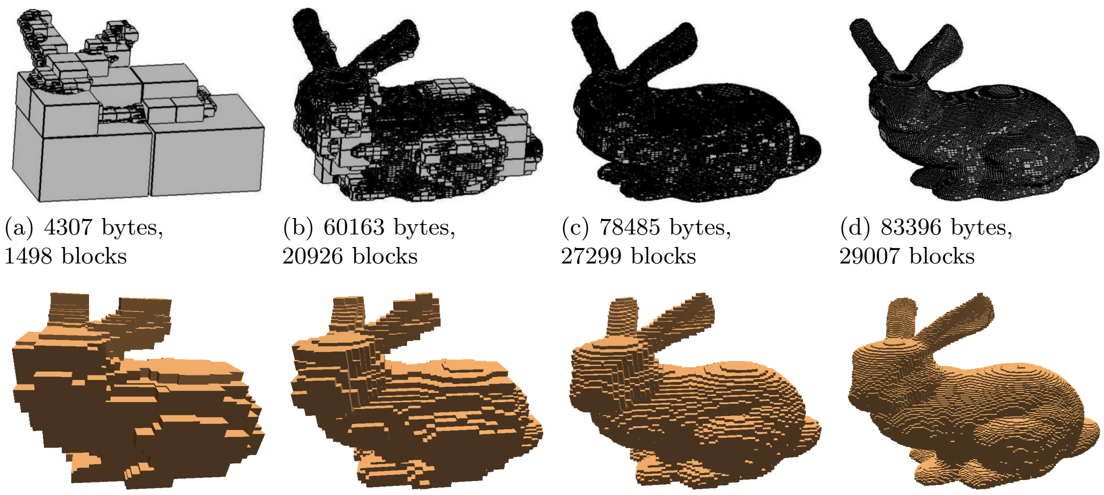

(a) 4307 bytes, 1498 blocks
(b) 60163 bytes, 20926 blocks
(c) 78485 bytes, 27299 blocks
(d) 83396 bytes, 29007 blocks
(e) 3880 bytes,2171 EV
(f) 8806 bytes,4879 EV
(g) 20796 bytes, 11429 EV
(h) 51698 bytes, 33298 EV

Figure 5.18: Bunny dataset. (a)-(d): Four objects generated with the octree-based method[1]. (e)-(h): Four objects generated with our approach.

[1]Figures featured in paper [135].

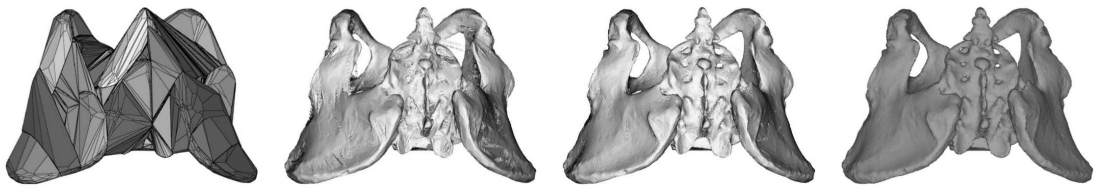
storage size of the octree-based method.

On the other hand, we have considered the streams (in plain text) generated by both methods and then we have zipped them. For the octree-based method 29007 random numbers between 1 and 4782968 have been generated, and for PEVE, a file like that shown in Figure 5.16 has been generated. In this case, the total storage size of the zipped data stream of the octree-based method is 107162 bytes[2]. For PEVE, the zipped data stream requires 87107 bytes. Therefore, this storage size is 0.812 of the storage size of the octree-based method.

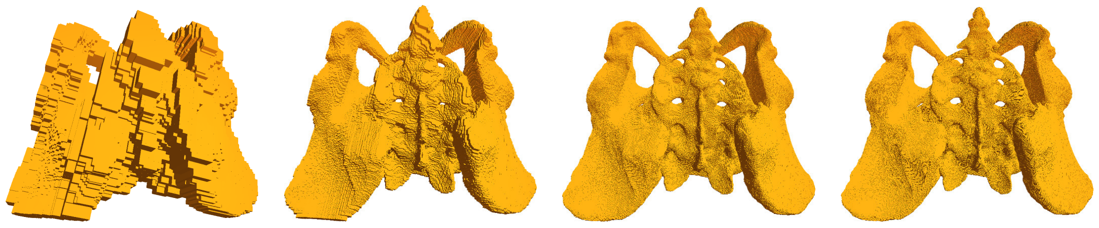
Figure 5.18 depicts a visual comparison of the approximation quality of both methods. For each object, it shows the quantity of decoded (transmitted) bytes and the number of elements (blocks and EV respectively). In all the approximations PEVE gives a better indication of the shape with less storage cost, and even EVM objects with 20796 and 51698 bytes look better than object with 60163 bytes in the octree-based method (see Figures 5.18(b), 5.18(g) and 5.18(h).

5.7.2 Comparison with a BSP-based Method

The BSP-based method of Huang and Wang [61] (BSP, from now on) generates approximations that are convex bounding volumes. Although this simplification method is progressive, it is lossy and does not allow a progressive decoding, so, separated approximations have been compared. Three datasets tested with BSP have been used: Pelvis, Venus and Foot (see Figures 5.19, 5.20 and 5.21). The basic element used in BSP is a plane represented by its 4 coefficients while for EVM is an EV represented by its 3 coordinates. Therefore, these datasets have been converted



(a) ≈7605 nodes (0.5%) (b) ≈456285 nodes (30%) (c) ≈760475 nodes (50%) (d) 1520949 nodes (original)

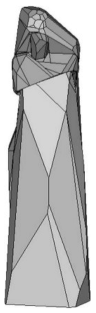


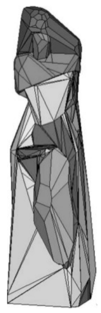
(e) 12030 EV (0.5%) (f) 130102 EV (6.2%) (g) 586502 EV (28.1%) (h) 2082078 EV (orig.)

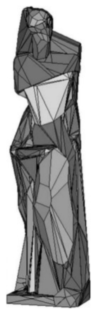
Figure 5.19: Pelvis dataset. (a)-(d): Four objects generated with the BSP-based method[3]. (e)-(h): Four objects generated with our approach.

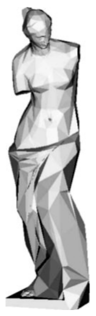
[2]Similar results for more than 20 tests.

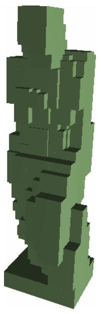
[3]Figures featured in paper [61].

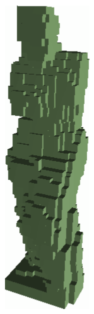

(a) ≈584 nodes, (5%)

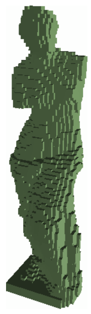

(b) ≈1167 nodes, (10%)


(c) ≈3502 nodes, (30%)


(d) 11673 nodes, (original)


(e) 612 EV (3.9%)


(f) 1360 EV (8.6%)


(g) 4304 EV (27.5%)

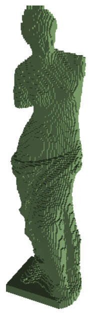
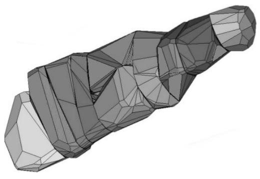
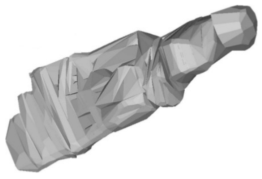
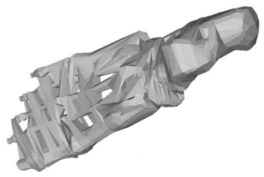
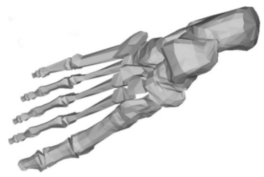

(h) 15636 EV (orig.)

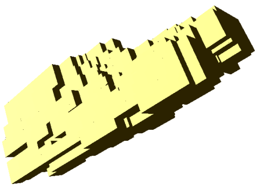
Figure 5.20: Venus dataset. (a)-(d): Four objects generated with the BSP-based method[2]. (e)-(h): Four objects generated with our approach.

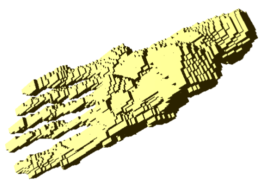

(a) ≈771 nodes, (2%)

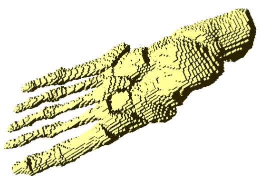

(b) ≈3854 nodes, (10%)


(c) ≈11561 nodes, (30%)


(d) 38535 nodes, (original)


(e) 630 EV (1.2%)


(f) 4206 EV (8.1%)


(g) 15324 EV (29.7%)

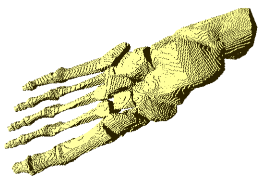

(h) 51532 EV (orig.)

Figure 5.21: Foot dataset. (a)-(d): Four objects generated with the BSP-based method[4]. (e)-(h): Four objects generated with our approach.

[4]Figures featured in paper [61].

into EVM in such a way that the number of EV is approximately 4/3 the number of planes of the initial object.

Table 5.4 gives the run times to obtain the LOD sequences in both methods. BSP has been tested with C++ program on a PC with Intel®Core 2 Quad CPU Q6600@2.40GHz. Note that although this PC is faster than the one we used, the run times with BSP are higher. Figures 5.19, 5.20 and 5.21 show several approximations using BSP and PEVE for the three datasets showing for each approximation the number of nodes (BSP), EV (PEVE) and the percentage of these elements with respect to the initial object.

Table 5.4: Run time (in seconds) for the BSP-based method and PEVE.

Dataset	BSP	PEVE
Venus	3.30	0.92
Foot	12.50	4.79
Donna	2382.00	471.30

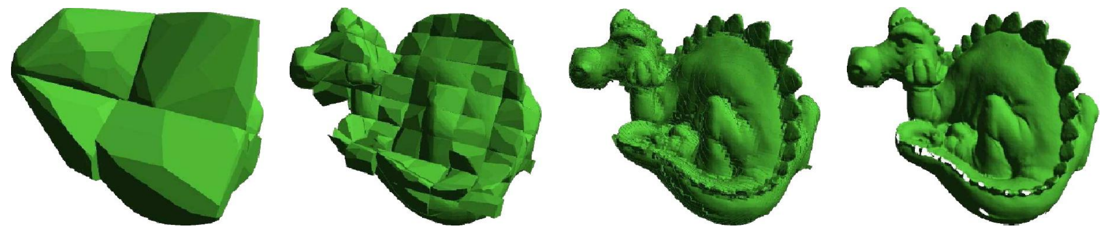
Concerning visual quality several observations can be made. For the Pelvis dataset, PEVE approximations maintain better the genus (number of handles), the genus of the original Pelvis is 13, and this value is maintained in the 28.1% approximation (see Figure 5.19(g)), the genus for the 6.2% and 0.5% approximations are 10 and 3 respectively. For the Venus dataset, PEVE approximations maintain better the concavities of the model than BSP. Finally, for the Foot dataset, the fingers in PEVE are noticeable even in the 1.2% approximation (see Figure 5.21(e)) while in BSP they look wrapped from the 30% approximation (see Figure 5.21(c)).

5.7.3 Comparison with the BP-Octree Method

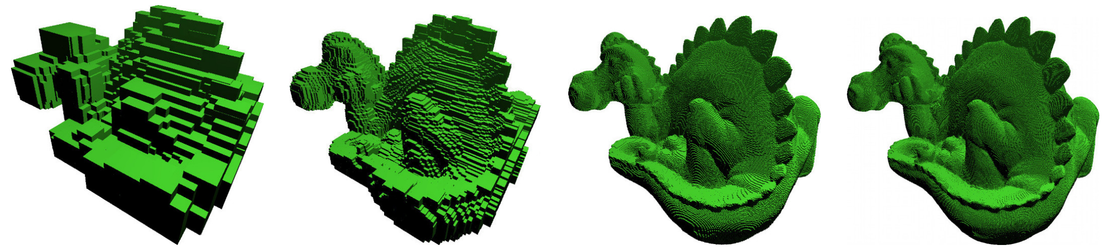
Melero et al. have tested the BP-Octree (BP-O) [94] to transmit progressively 3D objects over the network. The PhlegmaticDragon dataset has been tested, which has 715933 triangles (13298.9 kB in the PLY format). The generated BP-O uses a tree with a maximum depth of 8, i.e., it corresponds to a LOD sequence with 9 objects. It contains 970985 planes and supposes ≈25000 kB of transmitted data (1.8 times the original size). The authors affirm that the final geometry is never transmitted as a very fine approximation of the model is reached at leaf nodes. However, if the original geometry was transmitted (true lossless), this size increases.

On the other hand, the original PhlegmaticDragon dataset has been represented in EVM with 716840 EV (1309.9 kB, using the compressed EVM). Moreover, PEVE needs 1972.9 kB (1.5 times the original size) to encode all the 15 generated objects. Note that the required storage size in PEVE is noticeably smaller than using BP-O (7.8%) and that in PEVE the original geometry is always transmitted without any extra cost.

Figure 5.22 compare the visual quality of the shape approximation of BP-O and PEVE. For each approximation, it shows the percentage of decoded (transmitted) elements (planes and EV respectively) with respect to the original object as well as the transmitted kB. In all approximations PEVE gives a better indication of the shape with notably less storage cost. Note that the BP-O approximation looks wrapped in low levels. Finally, although the authors



(a) 2909 planes, (0.3%) ≈ 74 kB | (b) 26603 planes, (2.7%), ≈ 684 kB | (c) 403472 planes, (41.6%) ≈ 10388 kB | (d) 970985 planes, (100%) ≈ 25000 kB

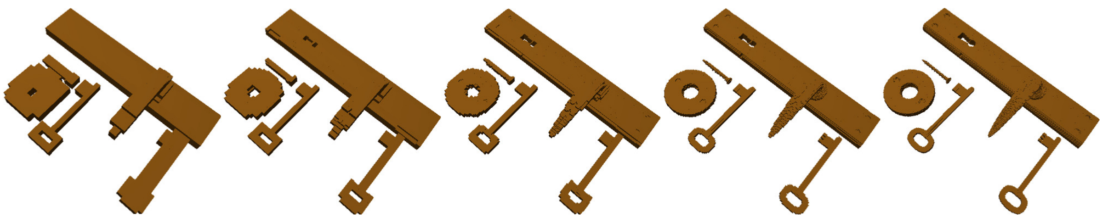


(e) 3412 EV, (0.3%) ≈ 8.3 kB | (f) 29140 EV, (4.0%) ≈ 86.9 kB | (g) 318661 EV, (34.0%) ≈ 782.8 kB | (h) 955901 EV, (lossless) ≈ 1972.9 kB

Figure 5.22: PhlegmaticDragon dataset. (a)-(c): Four objects generated by the BP-O method[5]. (d)-(f): Four objects generated by our approach.

do not indicate the PC used to generate the BP-O, they report the time it requires, 197.24 seconds, which is higher than to generate PEVE, 110.8 seconds.

More examples of approximations with PEVE are shown in Figures 5.23 to 5.28, indicating on each object the percentage of decoded bytes in order to get the original (lossless) object.



865 B (2.6%) | 1937 B (5.9%) | 5571 B (16.9%) | 13415 B (40.8%) | 32895 B (100%)

Figure 5.23: Some BOPP of the DoorPieces dataset.

[5]Figures featured in paper [94].

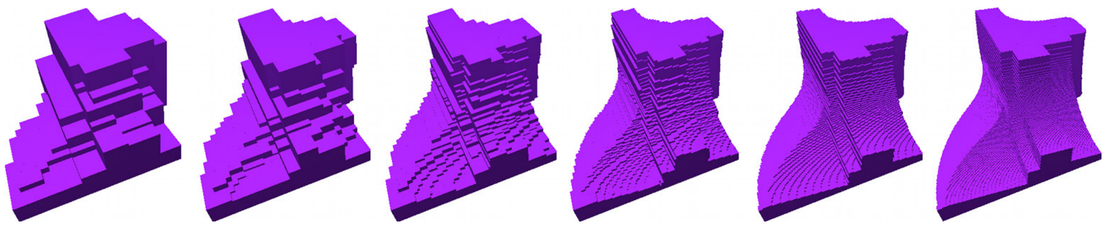


Figure 5.24: Some BOPP of the FanDisk dataset.

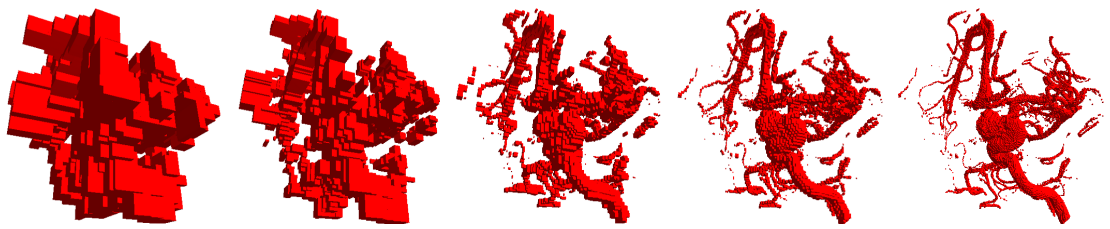


Figure 5.25: Some BOPP of the Aneurysm dataset.

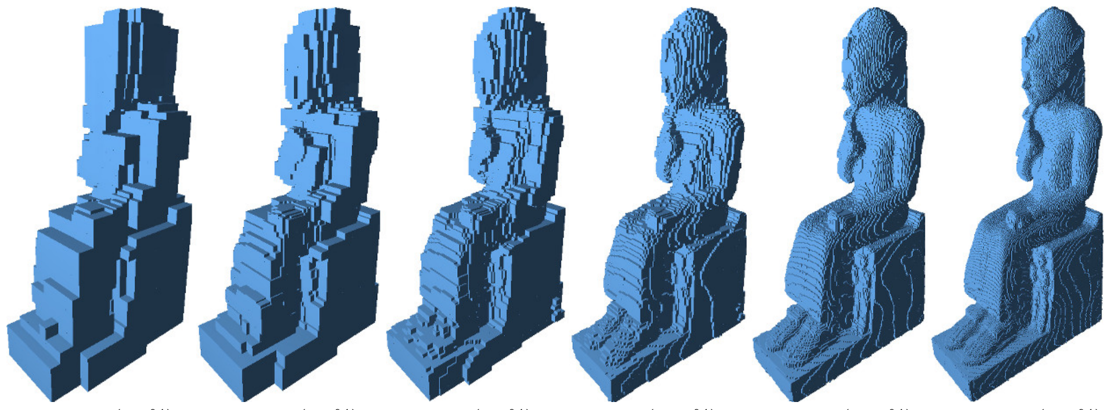


Figure 5.26: Some BOPP of the Ramesses dataset.

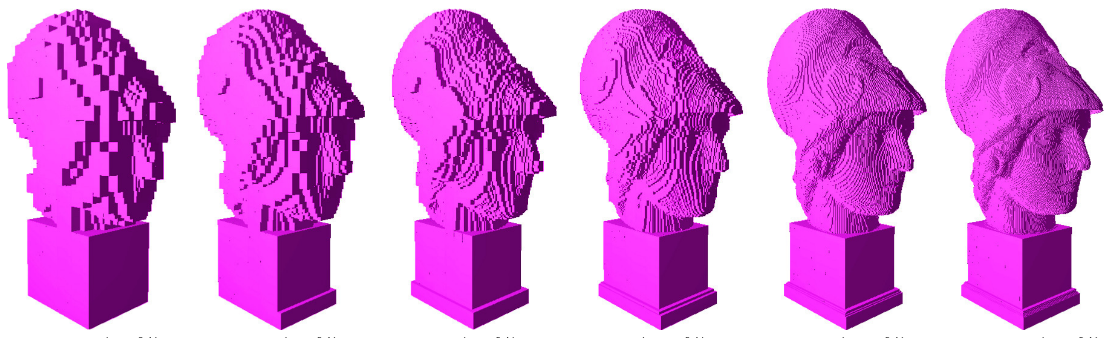


Figure 5.27: Some BOPP of the Athene dataset.

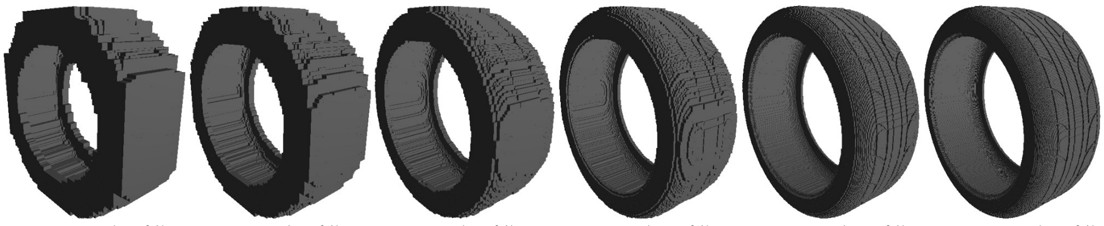


Figure 5.28: Some BOPP of the Tire dataset.

5.8 Simplification and Structural Parameters

In this section, experimental tests of the application of the simplification method as support in the structural parameters computation are presented and discussed. The objective is to show that the proposed simplification method not only can be used in classical applications such as collision detection, but also in the structural parameters computation.

5.8.1 Simplification and Pore-Size Distribution

Regarding the pore-size distribution, our hypothesis is that an approximation of a porous sample, generated with the proposed simplification method, can retain enough relevant information and, as the approximation will contain less number of basic elements (EV in EVM and boxes in CUDB) than the original dataset, it will be processed faster. This fact can be useful in several situations. For instance, suppose that we have a large number of samples, and we want to get a previous idea of their pore-size distribution in order to select the more outstanding samples, which will be further analyzed in more detail. Then, to prove this hypothesis, the pore-size distribution of all datasets depicted in Figure 4.13 have been computed, comparing the corresponding histograms and run times for the original object and for two approximations.

The approximations of each dataset have been computed using Algorithm 16 with $d_{max} = 1$ and $d_{max} = 2$, respectively. Table 5.5 shows the run times to compute the pore-size distribution of each dataset and its approximations. Note that the time required by the approximations is less than the time required by the original object in all cases, even considering the time

Table 5.5: Time required to apply MIP to the original object and to each approximation (MIP), and time required for the simplification (Simp) and total time (Total).

Dataset	Original	$d_{max} = 1$			$d_{max} = 2$		
	MIP	MIP	Simp.	Total	MIP	Simp.	Total
Scaffold	2.0	0.4	1.2	1.6	0.1	1.7	1.8
Trabecula	11.8	4.0	2.4	6.4	1.5	4.4	5.9
Glass	3.9	1.4	2.1	3.5	0.7	5.0	5.7
Stone	39.7	9.9	13.9	23.8	1.6	21.2	22.8
Soygel	446.2	38.0	155.4	193.4	12.6	228.6	241.2
Tween	230.6	63.2	28.0	91.2	26.2	54.8	81.0
PLA	271.7	55.9	63.0	118.9	22.4	111.6	134.0

required to compute the corresponding approximation for $d_{max} = 1$. For $d_{max} = 2$ this is also true except for the Glass dataset. But this dataset and Scaffold are the datasets showing the smaller run times. For the remaining five datasets the total run time is less than 60% of the run time of the original object. Finally, note that the total run times for $d_{max} = 1$ are similar to those for $d_{max} = 2$ and therefore we can conclude that the approximation corresponding to $d_{max} = 1$ could be a good choice.

All the generated histograms show a similar behavior. Here, the histograms of two of the biggest datasets (Soygel and Tween) are presented (see Figures 5.29 and 5.30). From the analysis of these histograms, it can be noted that curves corresponding to the original object and the approximations are very similar, specially with those corresponding to $d_{max} = 1$. It can also be noted that, the more simplified is the object the more pores information is lost; however, as expected, with the first approximation ($d_{max} = 1$), the objects retain enough relevant information and are processed more efficiently.

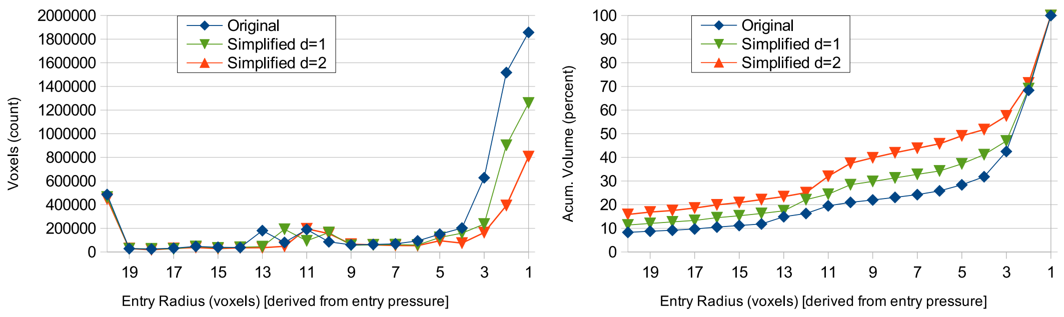


Figure 5.29: Absolute (left) and relative (right) pore-size distribution for the Soygel sample.

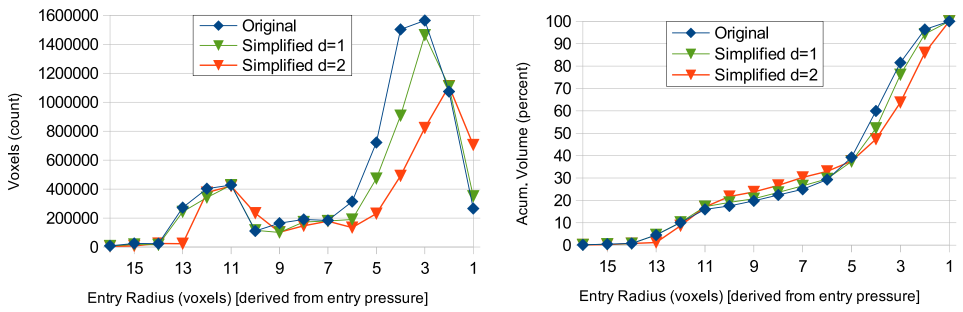
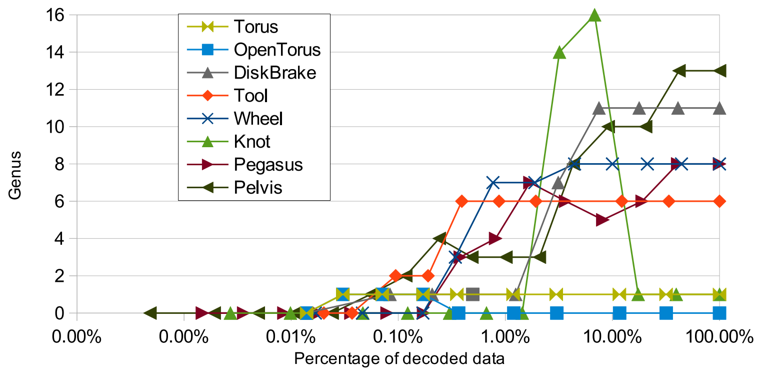


Figure 5.30: Absolute (left) and relative (right) pore-size distribution for the Tween sample.

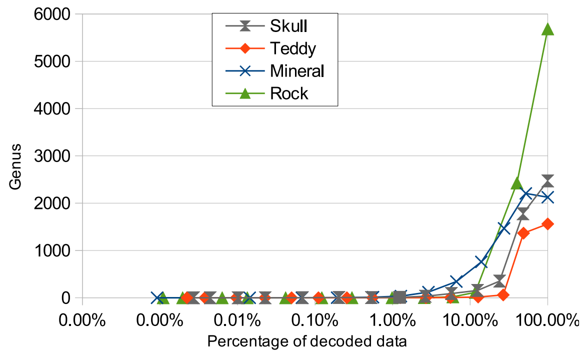
5.8.2 Simplification and Connectivity

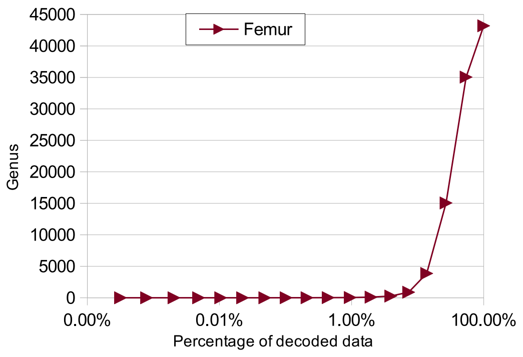
Although connectivity preservation is not an objective of the simplification method, it has been experimentally observed how the simplification alters the genus. Our hypothesis is that, datasets containing big holes will better preserve its genus than those datasets containing very small holes. To prove this hypothesis, some datasets depicted in Figure 4.20 have been simplified using Algorithm 17 with $n = 8$, i.e., until the AABB is obtained, and the genus of each approximation has been computed.

Figure 5.31 shows the genus as a function of the percentage of total bytes required to encode the LOD sequence. Each point in the curves corresponds to an approximation and the graphs are drawn in a logarithmic scale to show more details in the first approximations. Figure 5.31(a) corresponds to datasets with few and big holes, while Figure 5.31(b) corresponds to datasets with a lot of small holes. Note that, in the former, the genus is preserved in all datasets until approximations with 50% of encoded data, in some cases until 10%, and in the Tool dataset even until 1%. Observe that the genus of the Knot dataset takes a leap below 30% of encoded data. This is because as the model becomes coarser, the discretized surface of the object is self-intersecting producing artificial tunnels. Something similar occurs with the Pegasus and Pelvis datasets. In order to show simple cases, a torus and an open torus (C-shaped object) have been tested. The torus has a genus of 1 and this value is preserved in all but the last approximation (the AABB). In the case of the open torus, it has initially a genus of 0, but as it is simplified, it is reattached increasing its genus to 1.



(a)





(b)

Figure 5.31: Genus values as a function of the percentage of total bytes decoded.

On the other hand, note in Figure 5.31(b) that the genus of datasets with a lot of holes quickly decreases. This is because several holes measure one or two voxels in any of their dimensions, which implies that they are closed in the first approximations. Therefore, after this experimental analysis, it can be concluded that, the bigger the holes, the better the genus is maintained throughout the simplification.

5.8.3 Simplification, Sphericity and Roundness

Regarding the sphericity and roundness indices, it has been experimentally observed how the simplification alters this values. As the sphericity is most dependent on elongation, whereas roundness is dependent on the sharpness of the edges and corners, our hypothesis is that sphericity is better preserved than roundness because the simplification method keeps the AABB size in all the approximations, but the edges and corners tend to disappear. To prove this hypothesis, all the rock samples depicted in Figure 4.26 have been simplified using Algorithm 17 with $n = 8$, computing the sphericity and roundness indices of each approximation. All the samples show a similar behavior. For clarity purposes, graphs for seven samples (see Figure 5.32) are presented. They are representative of the distribution of the sphericity and roundness indices.

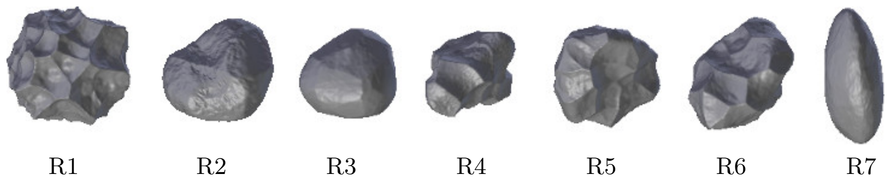


Figure 5.32: Rock samples tested with the simplification method.

Figures 5.33 and 5.34 show the sphericity and roundness indices respectively, as a function of the percentage of total bytes required to encode the LOD sequence. Each point in the curves corresponds to an approximation and the graphs are drawn in a logarithmic scale to show more details in the first approximations. Note in Figure 5.33 that, the sphericity index changes slightly in all samples until approximations with 10% of encoded data, and in some samples (R2, R3 and R7) until 1%. On the other hand, note in Figure 5.34 that, as expected, the

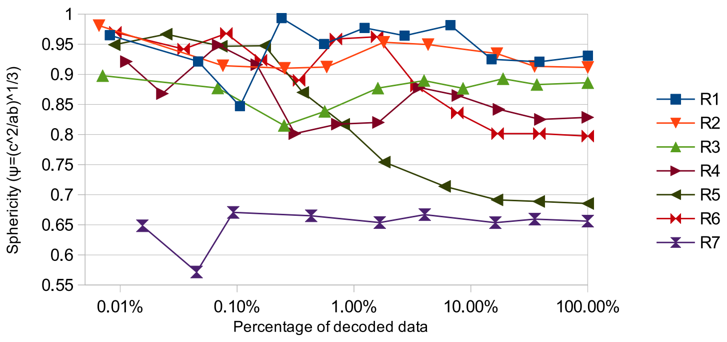


Figure 5.33: Sphericity

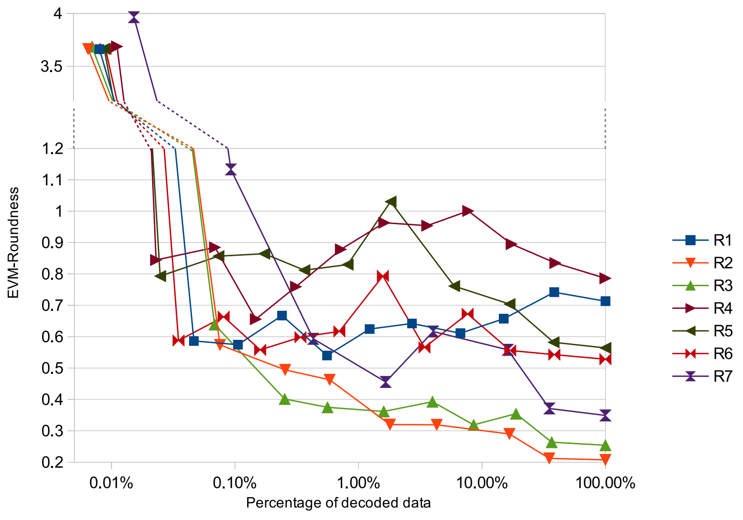


Figure 5.34: Roundness

roundness index is not well preserved. However, for the first approximations (around 50% of encoded data), it is almost the same that the original sample in all cases.

The variability of the values of the sphericity and roundness indices in the last approximations is due to the fact that sample's shape becomes increasingly coarse in block form. Therefore, after this experimental analysis, it can be concluded that the simplification method preserves the sphericity index until approximations with 10% of decoded data, but the roundness index is similar only for the first approximation.

5.9 Conclusions

This chapter has presented an approach to simplify orthogonal polyhedra and 3D binary images represented with EVM. It generates a LOD sequence of BOPP, which fulfills the common subset and AABB properties of bounding structures. The approach is based on a *merging faces* strategy and deals with OPP with any number of holes and connected components. It also includes a technique, based on model continuity, for a better shape preservation. Besides, the method takes full advantage of the efficiency and robustness of EVM Boolean operations, especially XOR. Finally, a data structure, PEVE, to encode in a progressive and lossless way the generated LOD sequence has been presented.

From experimental results, it can be concluded that the LOD sequence generated by the proposed simplification method is satisfactory in terms of quality of the approximation, although better quality is obtained when more orthogonal is the object's shape. Compared with other approaches, PEVE gives good compression rates and run times. Moreover, the generated bounding approximations are tighter, in general, than in other approaches, maintaining concavities, genus and other shape features at a very compressed size.

Finally, experimental tests have shown that the proposed simplification method can be employed as support in the structural parameters computation, since simplified objects provide enough relevant information and they have faster run times.

CHAPTER 6

Practical Applications

6.1 Introduction

This chapter presents two practical applications developed as part of multidisciplinary and collaborative research. Both applications compute structural parameters of datasets coming from real samples and rely on geometric techniques.

The first application (see Section 6.2) is a customized software analysis tool, which uses digital image processing techniques to automatically detect characteristic viscosity points (CVP) from a collection of 2D images of the sintering process of basalt or rock samples. This tool was developed to help determine the temperatures corresponding to CVP together with the hot-stage microscopy technique.

The second practical application (see Section 6.3) is a software that computes the sphericity and roundness indices of a real silica sand dataset coming from nano CT.

6.2 Hot-stage Microscopy Tool

6.2.1 Introduction

Hot-stage microscopy (HSM) is an analytical technique which combines the best properties of microscopy and thermal analysis to enable the characterization of the physical properties of materials as a function of temperature [173]. Viscosity-temperature curves can be obtained by combining HSM with a dilatometric analysis. The values corresponding to the morphological changes that occur during heating, and the different stages or physical changes that occur in the material in the process, make it possible to obtain accurate information on the behavior of the materials when they are subjected to a heating cycle.

During the experimental sintering process of a sample, a camera can take a picture at a certain interval of time, getting at the end a serie of images that depicts the evolution of the sample as it melts over time. CVP corresponding to Scholze's definition [142] of this process evolution are as follows:

1. First Shrinkage: Shrinkage starts.

2. Maximum Shrinkage: Before the material starts to soften.
3. Softening: Softening starts.
4. Ball: The material forms a sphere.
5. Half ball: The material forms a semi-sphere.
6. Flow: The material is fused.

6.2.2 Input and Preprocessing

Every dataset consists of a collection between 300 and 700 grayscale TIFF images of 2048×1536 and 16 bits each one. Figure 6.1(a) shows one of these images. Each collection depicts the evolution over time of the melting in the furnace of a sample. Associated with each collection there is the applied temperature increment in °C/min. and a list of temperature ranges, where for each range the following information is provided: the frequency (images/min.) at which the pictures were taken, the corresponding first image index and the initial temperature of the range. All this information allows to tag each image with the furnace temperature at the time it was taken.

In order to obtain the desired measurements, the image needs to be preprocessed first. This preprocessing basically consists of a binary threshold filter, a noise reduction filter and an alignment operation.

Binary Threshold Filter

This filter reduces the gray scale image to a black and white one. An automatic threshold value is computed in run time taking into account the arithmetic mean and the standard deviation of the different gray values (see Figure 6.1(b)). However, if images have a poor contrast, the computed threshold could not be appropriated to segment them correctly. Therefore, this parameter can be defined by the user.

Noise Reduction

The previous operation can generate salt and pepper noise. There exist several image processing filters that can be applied to reduce the noise. As tiny parts are considered noise instead of features (see Figure 6.1(c)), a classical morphological opening and closing operations are applied to the binarized image.

Alignment and Clip Operation

Images can be sloped because the camera may have been misaligned. In order to correctly align them, the pixel column in the right half of the image and the pixel column in the left half closest to the sample, which contain the highest number of white pixels, are detected. Then, the correct angle to rotate the image is computed and applied to the image. Moreover the excess floor is clipped (see Figure 6.1(d)).

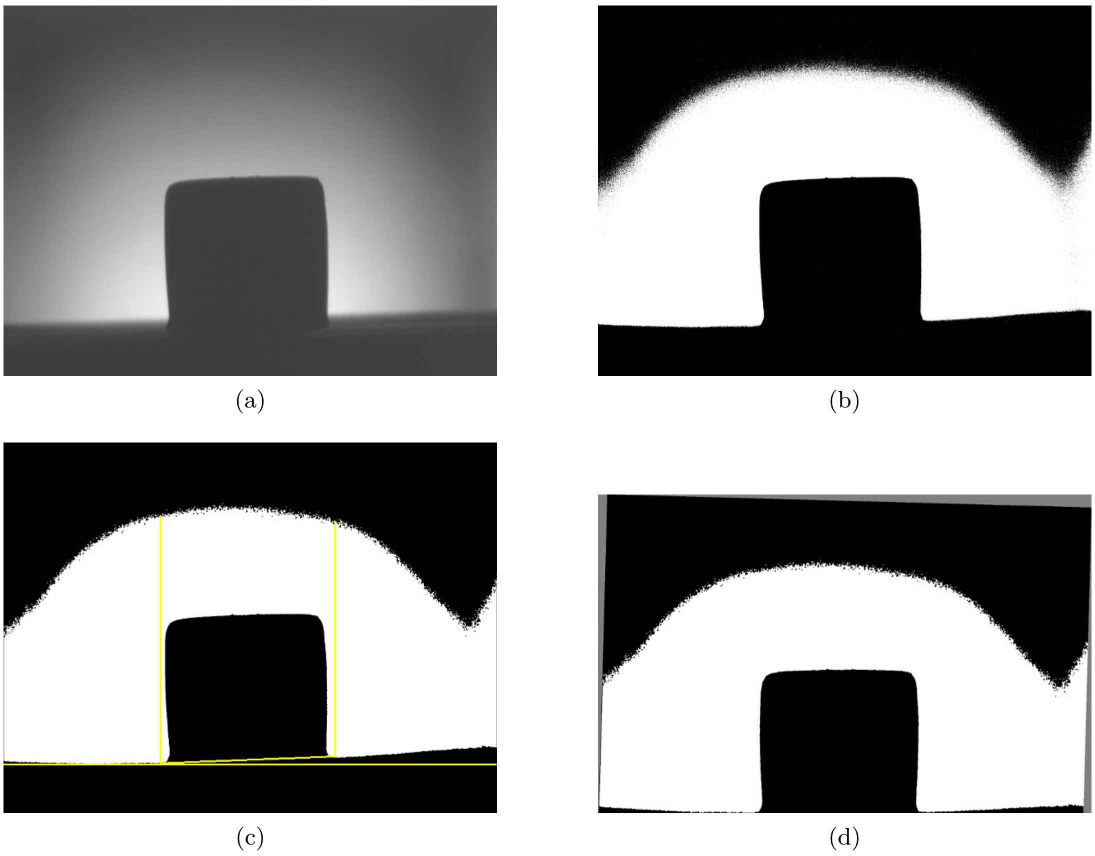


Figure 6.1: Example of a captured basalt image. (a) Original image. (b) After binary threshold filter. (c) After noise reduction (Observe that the image is not aligned) (d) After alignment operation.

6.2.3 Distinctive Data Detection

After the preprocessing, a segmentation is required to separate the sample from the background. In order to do this some *distinctive data* are detected first.

Distinctive Data

Distinctive data is the set of variables that help to calculate the required parameters to obtain the CVP. These variables are (see Figure 6.2):

V_0	Pixel column in the left half closest to the sample with more white pixels.
V_1	Pixel column in the right half closest to the sample with more white pixels.
$\mathbf{a}$	Point in the left-top corner of the sample.
$\mathbf{b}$	Point in the right-top corner of the sample.
$\mathbf{c}$	Point in the left-bottom corner of the sample.
$\mathbf{d}$	Point in the right-bottom corner of the sample.
w_{max}	Length of the sample's longest row.
h_{max}	Length of the sample's longest column.

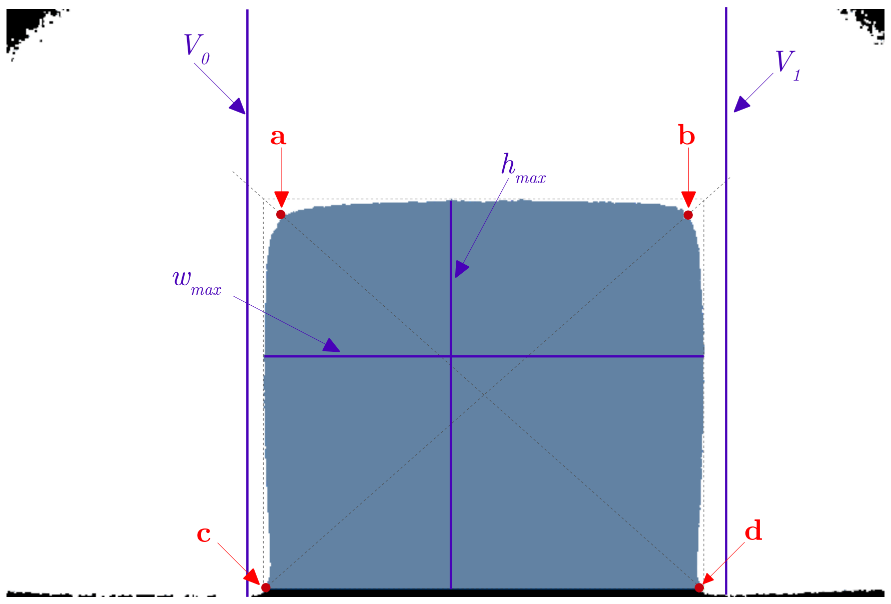


Figure 6.2: Segmentation and distinctive points

area	Area of the sample.
perimeter	Perimeter of the sample.

Points **c** and **d** correspond to the position of the first white pixel that appears (at the bottom of the image) from the center to the left and right of the image respectively. These positions are used in a flood fill algorithm in order to get a segment image like that shown in Figure 6.2. After segmenting, w_{max} and h_{max}, as well as the *perimeter* and the *area* of the sample, can be computed.

Points **a** and **b** are not trivial to detect. These points are estimated based on the traces of lines from the points **c** and **d** to the corners of the segmented sample AABB. Then, the position of the first white pixels that these lines meet, correspond to the points **a** and **b**.

6.2.4 CVP Computation

Let $S = \{s_1, s_2, \ldots, s_n\}$ be the set of n images that depict the evolution over time of the melting in a furnace of a sample. For each image in S, the above described preprocessing is applied to compute all its *distinctive data*. After that, the set of segmented images is ready to detect the six CVP corresponding to Scholze's definition.

First and Maximum Shrinkage

First and maximum shrinkage are characterized by the sample's size starting to decrease and reaching its maximum decrease respectively. In order to detect these CVP, the evolution of the sample's area along time is plotted with a line chart. The common characteristic in these kind of charts is that the first shrinkage occurs approximately when the line begins to drop (see Figure 6.3), from here, the sample continues decreasing and when the line is flat again, the maximum shrinkage is reached. At the beginning of the melting process, the sample can inflate briefly, but, according to the experts, it generally occurs before 200°C. So, the first values of the chart under this temperature are ignored.

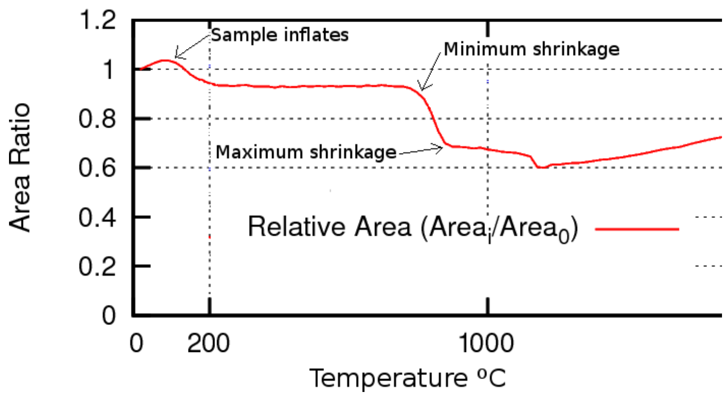


Figure 6.3: Graph of the area sample trough time.

To detect this CVP in the chart, the slope angle is computed at every point. When it is large enough, the image s_i at this point corresponds to the first shrinkage CVP. Then, when the slope angle tends to zero again, the image s_j at this point corresponds to the maximum shrinkage CVP. To increase measurement accuracy, the chart points are analyzed in groups of 3. Initially, an angle of 15° is considered by default as large enough to detect the curvature. However, this parameter can be defined by the user.

Softening

The softening point is characterized by the disappearance of sharp parts and the beveling of the corners of the sample (corners and peaks start to soften). Three different alternatives can be applied to obtain this CVP.

1. The first alternative gets the image s_i having the ratio between s_i and its AABB areas closest to 1.

$$\frac{s_i.area}{area(AABB(s_i))} \approx 1 \tag{6.1}$$

2. The second alternative gets the image s_i having the ratio between the line segment length from **a** to **d** (or from **c** to **b**) and the diagonal length of its AABB closest to a given threshold. This threshold is defined by the user and is initially set to 0.95.

$$\frac{length(s_i.\mathbf{a}, s_i.\mathbf{d})}{diagonal(s_i)} \approx threshold, \qquad \frac{length(s_i.\mathbf{c}, s_i.\mathbf{b})}{diagonal(s_i)} \approx threshold \tag{6.2}$$

3. The third alternative is based on the Freeman chain code[39]. The Freeman code with 8-connected grid (see Figure 6.4) is used to encode in a counter-clockwise sense the 10%

of the top of the image (see Figure 6.5(a)). Since the softening CVP is characterized by corners and peaks starting to disappear, the image with the most flat top is a candidate to represent this point. Therefore, the softening CVP is estimated as that image s_i having a chain code with the maximum ratio between the number of occurrences of 6-direction and the size of the chain. An example is shown in Figure 6.5(b).

$$max(countCCode6(s_i)/sizeCCode(s_i)) \tag{6.3}$$

3	4	5
2	*	6
1	0	7

Figure 6.4: Freeman code scheme with 8 directions.

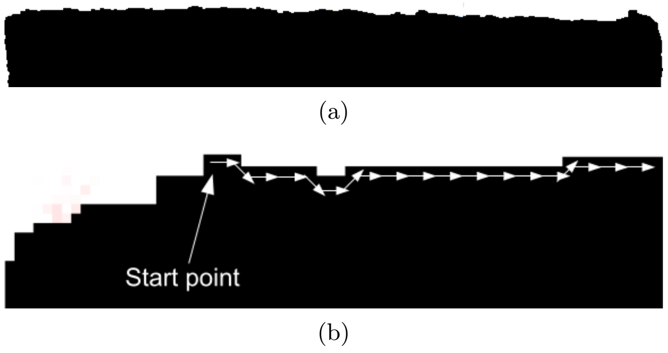


Figure 6.5: Example of Freeman chain code. (a) The top 10% of a sample. (b) Zoom and its chain code: 6766765666666665666...

Equations 6.2 and 6.3 were selected in the final application because they give a more accurate result for the softening CVP.

Ball

This point is characterized by the sample's shape being a ball. When the sample is close to a circular shape, it can be seen as a circle inscribed in a square. However, the sample actually never forms a perfect circle shape but a circular segment (see Figure 6.6(a)), where approximately the 15% of the circle height is lost.

Then, the ratio between the line segment defined by the left-bottom corner of AABB $\mathbf{p_0}$ and the point $\mathbf{b}$ (or by the right-bottom corner of AABB and $\mathbf{a}$) of length g, and the AABB diagonal of length d, is used to determine the ball CVP (see Figure 6.6(b)).

To determine g, a line-circle intersection is performed. The line-circle intersection equation for a line starting at point $\mathbf{p_0} = (0, 0)$ is defined as $|\mathbf{v}|^2t^2 - 2(\mathbf{v} \cdot \mathbf{o})t + |\mathbf{o}|^2 - r^2 = 0$, where $\mathbf{v}$ is the director vector of the line and $\mathbf{o}$ the center of the circle with radius r. In this case, $\mathbf{v} = (2r, 1.7r)$

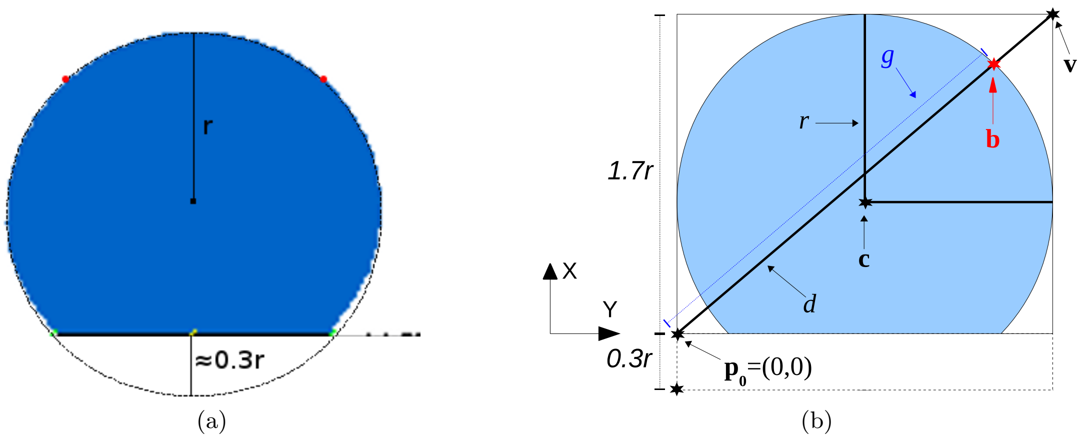


Figure 6.6: Circular segment. (a) A real sample shape. (a) Geometric representation.

and $\mathbf{o} = (r, 0.7r)$. Thus, the quadratic equation is $0.689r^2t^2 - 0.638r^2t + 0.049r^2 = 0$. Solving for t, $t \approx 0.81463$, $\mathbf{b} \approx (1.68292r, 1.43048r)$, and the desired ratio $g/d \approx 0.84146$.

Note that $\mathbf{b}$ corresponds to the point detected in the distinctive data. Therefore, the ball CVP is estimated as that image s_i having the ratio g/d closest to 0.84146.

$$\frac{length\_g(s_i)}{diagonal(s_i)} \approx 0.84146 \tag{6.4}$$

Half Ball

This point is characterized by the sample shape being a semi-circle. As this CVP appears after the ball CVP, it can be detected as the image s_i having a width twice its height.

$$\frac{s_i.w_{max}}{s_i.h_{max}} \approx 2 \tag{6.5}$$

Flow

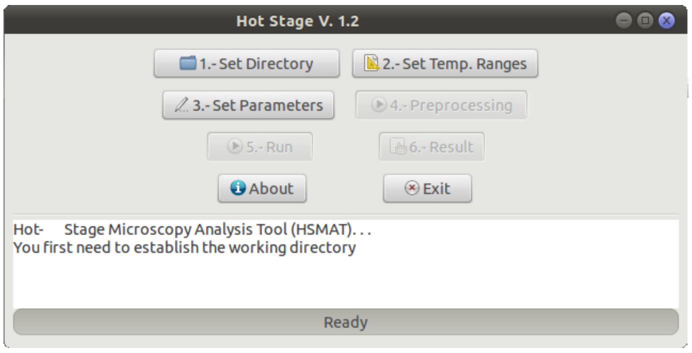
The las CVP is assigned to the image s_i having the ratio $s_i.h_{max}/s_1.h_{max}$ closest to a constant k. According to the experts, k is usually fixed between 10 and 16%.

$$\frac{s_i.h_{max}}{s_1.h_{max}} \approx k \tag{6.6}$$

6.2.5 Experimental Results

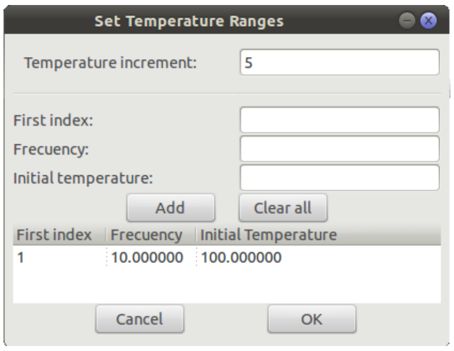
The software has been written in C++ using the GTKMM toolkit as its interface. It has been tested on a PC Intel®Core 2 Duo CPU E6600@2.40GHz with 3.2 GB RAM and running Linux. The software save all processed data for future analysis and allows users to modify some parameters such as the angle for the shrinkage CVP, and the thresholds for the binary filter and the softening and flow CVP points. Figure 6.7 shows screenshots of the developed software Hot-stage Microscopy Tool.

Hot-stage Microscopy Tool has been tested with several real datasets. Here, results for two glass samples obtained from wastewater treatment plants are presented. A dataset contains 369 images and the other one 667 images. Figure 6.8 shows the results, in these cases all of the CVP were detected using the default parameters.



Hot- Stage Microscopy Analysis Tool (HSMAT). . .
You first need to establish the working directory

(a)

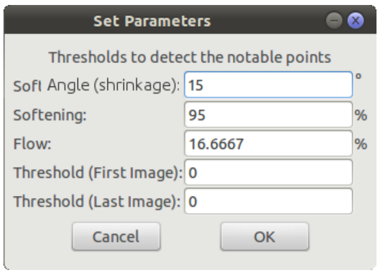


Temperature increment: 5

First index:
Frecuency:
Initial temperature:

First index	Frecuency	Initial Temperature
1	10.000000	100.000000

(b)

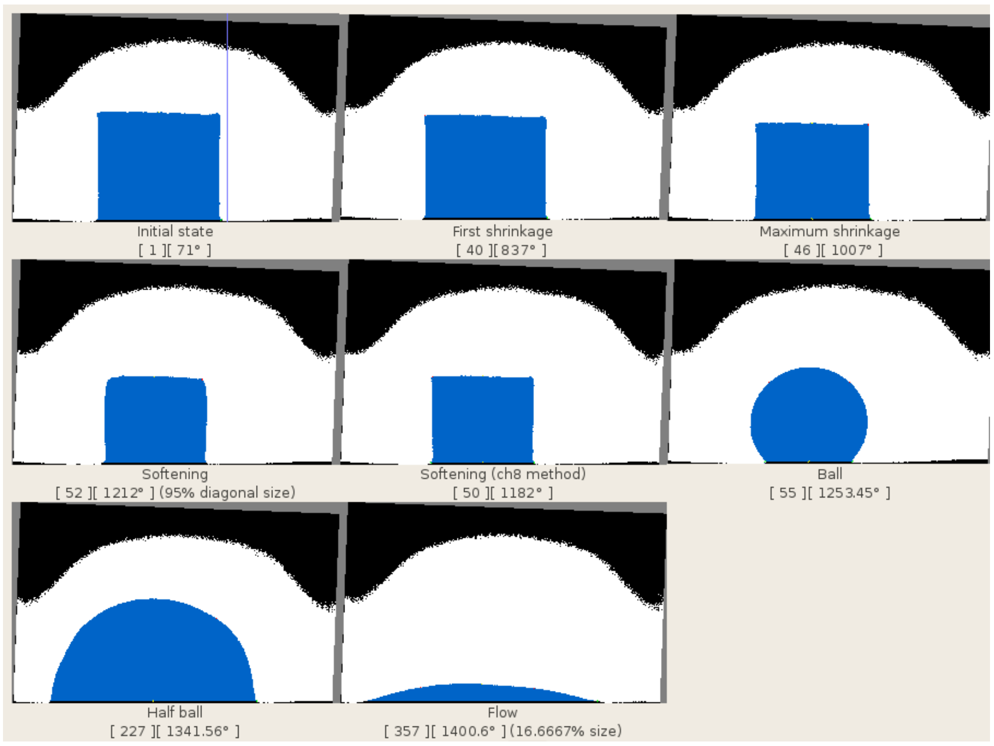


Thresholds to detect the notable points

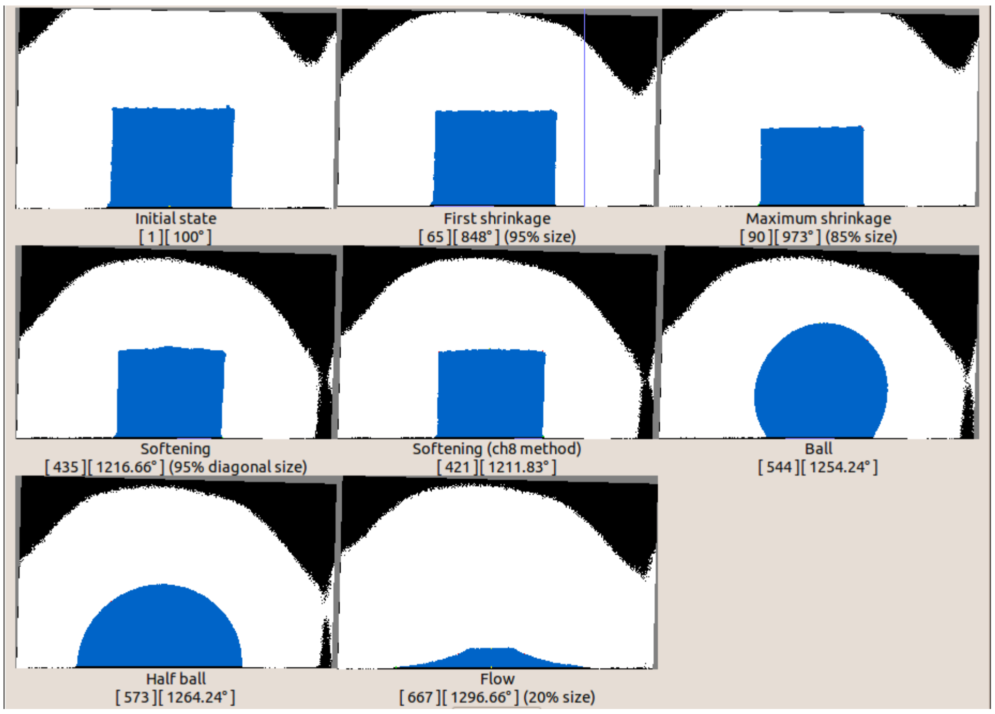
Sofl Angle (shrinkage):	15	°
Softening:	95	%
Flow:	16.6667	%
Threshold (First Image):	0	
Threshold (Last Image):	0	

(c)

Figure 6.7: Screenshots of Hot-stage Microscopy Tool. (a) Main interface. (b) Temperatures window. (c) Parameters window.



(a)



(b)

Figure 6.8: Results of Hot-stage Microscopy Tool for two glass samples obtained from wastewater treatment plants.

6.3 Analysis of Real Silica nano CT

6.3.1 Introduction

Silica sand is one of the most common varieties of sand found in the world. It is used for a wide range of applications. Sand consists of small grains or particles of mineral and rock fragments, where the dominant component is the mineral quartz, which is composed of silica (silicon dioxide). Sand having high silica levels and used for purposes other than the construction is referred as silica sand or industrial sand.

Industrial uses of silica sand depend on its purity and physical characteristics. Some of the more important physical properties are: grain size, grain shape (sphericity and roundness), grain strength and refractoriness. For instance, silica sand particles need to have a high sphericity and roundness, as the more round and spherical is the particle, the more resistant that particle is to crushing or fragmenting. In this section, we focus on the sphericity and roundness computation of the grains of a real silica nano CT.

6.3.2 Input and Preprocessing

We have been provided with a 32-bit gray scale dataset in RAW format with dimensions 131×281×332 of a silica nano CT. Some 2D slices of this dataset are depicted in Figure 6.9.

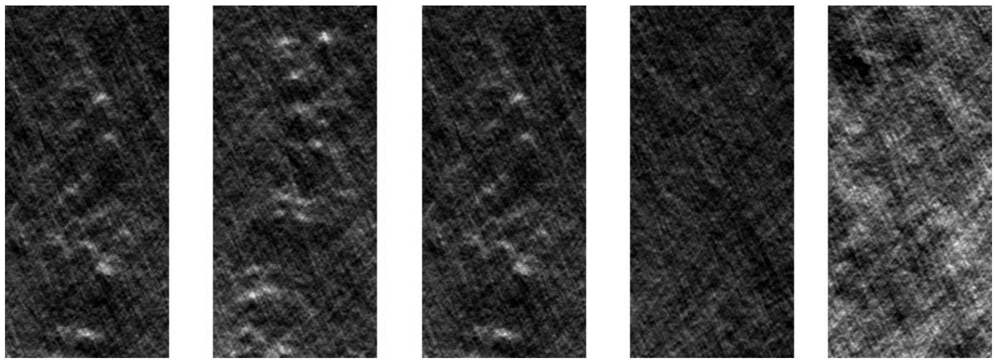


Figure 6.9: 2D slices of a silica nano CT.

In order to obtain the desired measurements, the dataset needs to be preprocessed. This preprocessing basically consists of the next steps:

1. Convert the original dataset to a 8-bit voxel model.
2. Scale the voxel model, via trilinear interpolation, to twice its size in order to better define the grain shapes.
3. Apply a binary threshold filter.
4. Remove noise applying morphological opening and closing operations.

Step 3 requires a threshold that allows to yield a good segmentation of the grain shapes. The gray value histogram is shown in Figure 6.10, in this case, high values of gray represent the foreground and the curve helps to determine where the background ends and the foreground begins. Note that the curve falls off around the value of 50. Therefore, thresholds of 45, 50 and 55 have been used to binarize the dataset. Figure 6.11 shows the sample after the corresponding threshold and noise removal, where for each threshold, the number of connected components (CC), considering 6-connectivity, is indicated. Observe that the result of applying a threshold of 45 looks like a model having several agglomerated grains. The result of applying a threshold of 55 seems to lose information. Therefore, we consider that the grain shapes are better defined applying a threshold of 50.

After the preprocessing, the voxel model is converted to its corresponding CUDB-representation in order to get the connected components, which represent the grain particles. Figure 6.12 shows a graph of the number of CC according to their volumes. CC having a very small volume are not representative of a grain particle. According to this graph, we consider as grains those

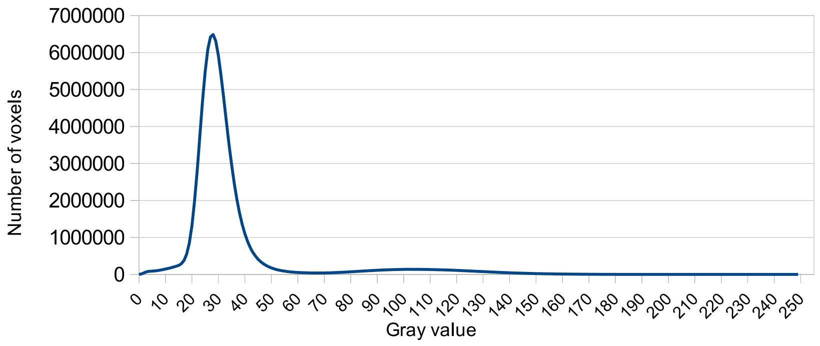
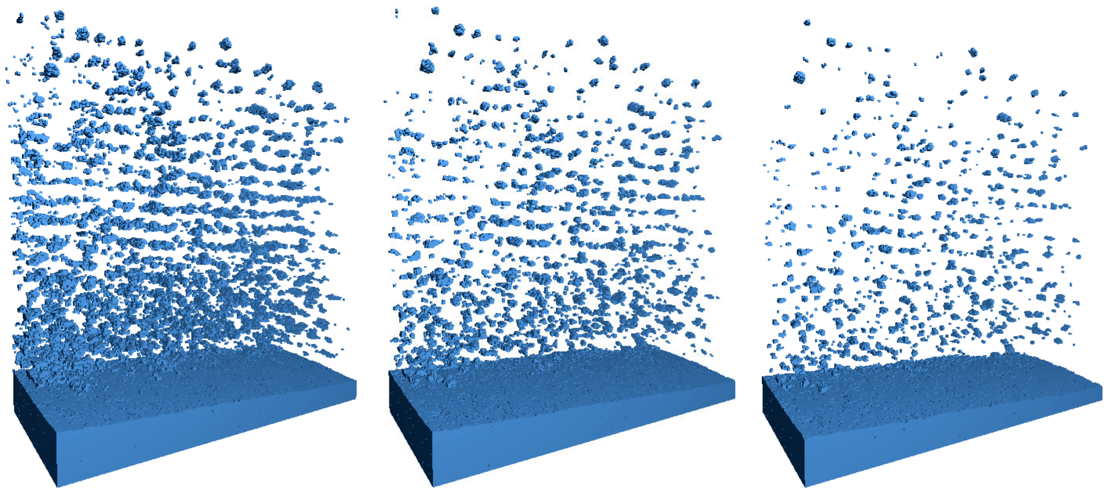


Figure 6.10: Gray value histogram of the sample.



(a) CC=1894 (b) CC=1181 (c) CC=753

Figure 6.11: Dataset after segmentation and noise removal with thresholds (a) 45, (b) 50 and (c) 55.

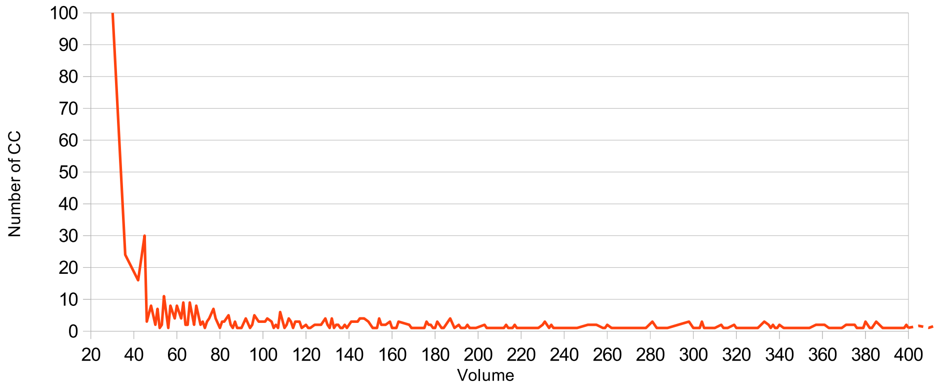


Figure 6.12: Number of connected components according to their volumes.

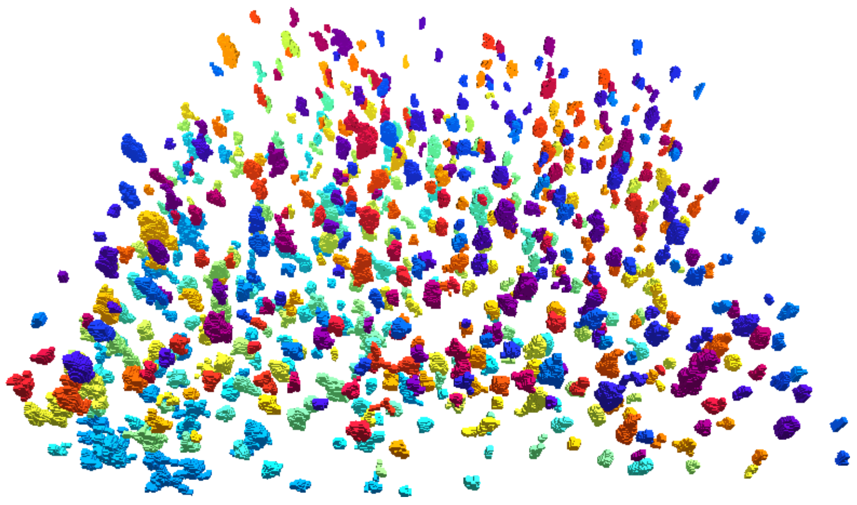


Figure 6.13: Resulting grain particles with volume larger than 200 voxels.

CC with a volume larger than 200 voxels (shapes larger than approximately 6×6×6). Therefore, the resulting dataset consists of 650 CC. See Figure 6.13 where the resulting grains are depicted.

6.3.3 Grain Properties Computation

After the grain particles have been correctly defined, they are converted to its EVM-representation in order to compute its sphericity and roundness indices.

Sphericity

The sphericity index of each connected component is computed using Equations 2.15 and 2.19. Figures 6.14 and 6.15 show the bar charts that represent the number of occurrences of the corresponding sphericity index. Although the voxel-based scheme used to compute the surface area (see Section 4.4.2) is not well suited for too small objects, both graphs are similar and give an estimation of the sphericity distribution.

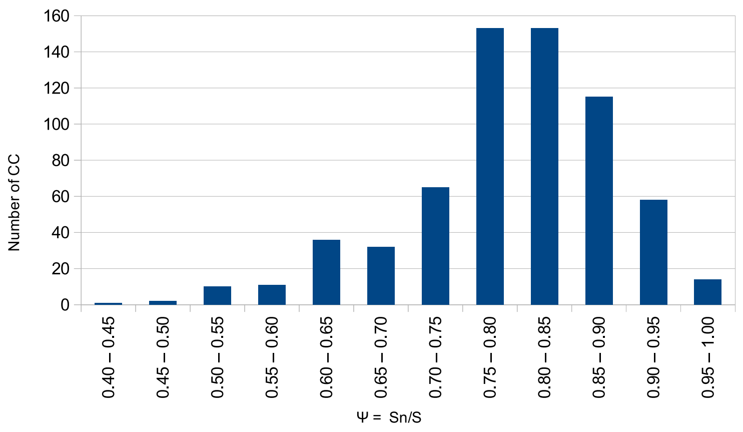


Figure 6.14: Occurrences of sphericity indices using Equation 2.15.

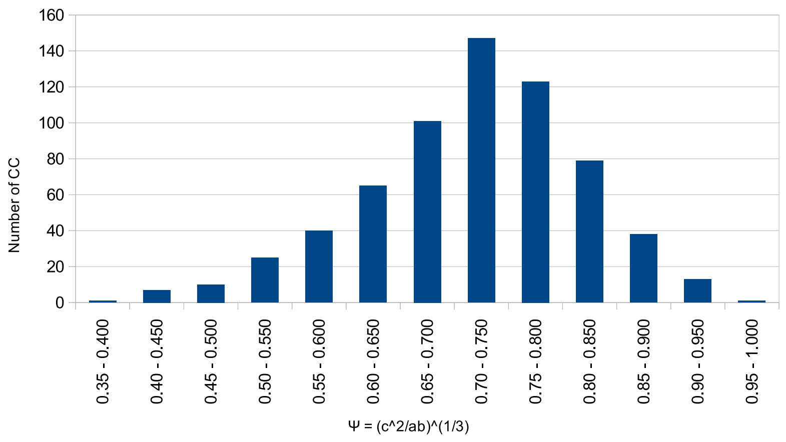


Figure 6.15: Occurrences of sphericity indices using Equation 2.19.

Roundness

Regarding roundness, for each connected component, its 3D roundness index is computed using Equation 2.21) and the EVM-based roundness (see Section 4.4.3). Figures 6.16 and 6.17 show the bar charts that represent the number of occurrences of the corresponding roundness index. Again, although the roundness indices are computed using different approaches, both graphs are similar and give an estimation of the roundness distribution.

6.4 Conclusions

This Chapter has presented two practical applications, which have been used as support by experts in order to extend their observations and help reach conclusions about the studied samples. Hot-stage Microscopy Tool has been satisfactorily used by experts to establish the relationship between the chemical composition, temperature and viscosity of diverse samples such as basalts rocks and glasses. Results of the second application have been used by experts in nanotechnology who are interested in segmenting and analyzing the sphericity and roundness of the silica nano sample presented here.

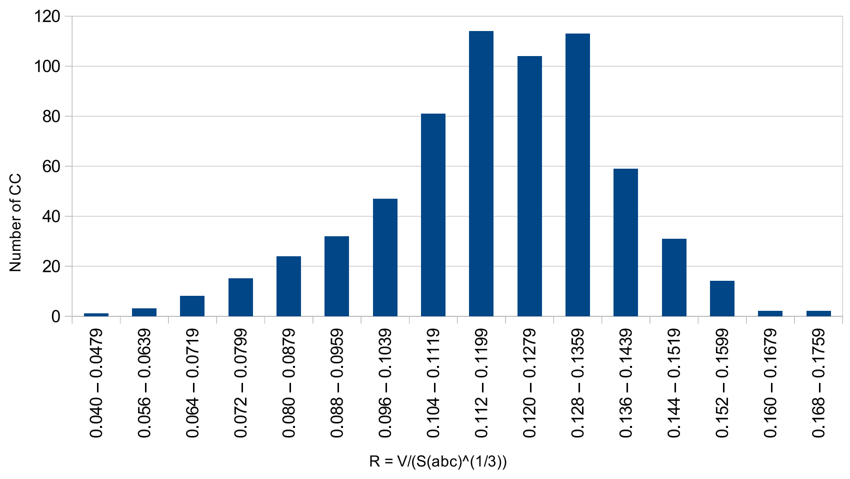


Figure 6.16: Occurrences of roundness indices using Equation 2.21.

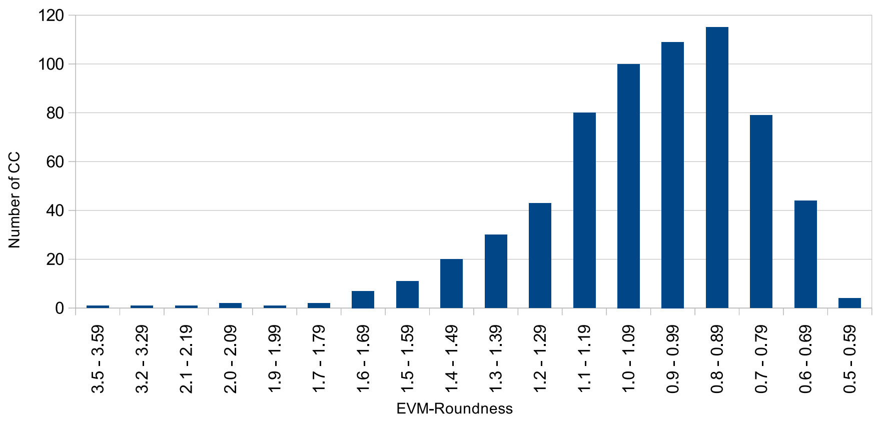


Figure 6.17: Occurrences of roundness indices using Equation 4.29 (EVM-roundness).

CHAPTER 7

Conclusions and Future Work

7.1 Conclusions

This thesis has proposed methods for the computation of several structural parameters of binary volume datasets coming from both phantom and real samples. The Extreme Vertices Model and a new proposed decomposition model, Compact Union of Disjoint Boxes (CUDB), have been used to represent the datasets and to compute structural parameters in an efficient and compact way. Moreover, a new EVM-based method to simplify binary volumes has been also presented. The main contributions of the thesis are summarized as follows:

- An improved decomposition model for OPP. Experimental results show that CUDB has several advantages compared with other decomposition models such as OUDB-extended, mainly its performance in methods such as CCL and collision detection used in the structural parameters computation.

- A CUDB-based virtual porosimeter, which simulate mercury intrusion at increasing pressures, like the porosimeter lab device. The main advantage of this method is its performance with respect to previous approaches as it does not require a prior computation of the skeleton.

- A method to compute the Euler characteristic and the genus of binary volumes. This method exploits the advantages of CUDB and is notably faster (up to two orders of magnitude) than previous approaches.

- Methods to estimate sphericity and roundness indices based on the computation of the object's OBB from EVM. Several sphericity indices have been computed from the OBB. An EVM-based roundness method based on a ray-casting-like approach has been presented, which shows good correlations with previous methods and is faster than them.

- A new approach to simplify binary images represented with EVM. It generates a LOD sequence of BOPP, which fulfill common properties of bounding structures. The method deals with OPP with any number of holes and connected components, and it encodes in a progressive and lossless way the generated LOD sequence. Compared with other approaches, the method gives good compression rates and run times, and the approximations are tighter, in general, maintaining concavities, genus and other shape features. Moreover, this simplification method can be employed as support in the structural parameter computation since the approximations provide enough relevant information and have a faster run time than the original object.

- Two practical applications used by experts. Hot-stage Microscopy Tool has been satisfactorily used to detect characteristic viscosity points of basalt and glass samples. Sphericity and roundness methods have been applied in a real silica nano CT.

The main advantage of the methods presented in this thesis to compute structural parameters is its performance with respect to previous approaches. On the other hand, real samples scanned at high resolution usually generate huge datasets that require a lot of memory and large processing time to permit their analysis. However, experimental test show that simplified objects generated by the proposed simplification method, provide enough relevant information while allowing faster run times.

7.2 Future Work

In this thesis, three different research fields related with binary volumes have been dealt: model representation, structural parameters computation and model simplification. Several activities can be proposed for each of these fields as future work:

- Development of the CUDB to B-Rep conversion. An interesting method would be the conversion from CUDB to B-Rep. We think it is possible to obtain a B-Rep model analyzing the boxes and its neighborhood.

- Analyze if any of the six ABC-sortings in CUDB produces an optimal number of disjoint boxes. CUDB represents objects with a small number of elements. However, if any ABC-sorting produces the smallest number of orthogonal disjoint boxes that cover the OPP has not been studied. Otherwise, one could think of allowing overlapping boxes in order to further reduce this number. Although this implies modifying the developed algorithms.

- Development of a method for pore space partitioning using a different approach to virtual porosimetry and avoiding the skeleton computation. The narrow throats detection presented in Section 4.2.3 can be applied to this method. If the lengths of all throats in the porous space are computed at once, they can be analyzed in order to detect the shortest paths that connect the solid space. Then, those paths that best split the porous space can be determined. In 2D this method is simple, since throats are represented as lines. However, in 3D case, the throats are represented as rectangles and the general oblique throat by three rectangles, which requires some thought in order to provide a robust technique.

- Improvement of the connectivity computation. In the proposed method, the object's complement is computed from EVM, therefore, conversions from CUDB to EVM and vice versa are required. The development of a method to compute the object's complement directly from its CUDB-representation will improve the run time of the connectivity computation.

- Development of a method that, from an EVM model, detects the surface voxels and classifies them into the nine classes defined by the voxel-based scheme [181] used in Section 4.4.2 to estimate the real surface area of an object. From the EVM to B-Rep algorithm [3], it can studied an approach that directly detects and classifies the surface voxels.

- Development of a variant of the simplification method, which preserves the connectivity, i.e., the genus. It implies defining a new treatment of the void space, where the faces that define the holes must be exhaustively analyzed in order to prevent them from closing completely.

- We have compared PEVE in storage and processing cost with methods that produce a progressive LOD sequence of bounding volumes. With the aforementioned variant of the method, the comparison can be extended to methods that use other lossless representation models for binary volumes such as run-length-encoding [117], ray representation [95] or triangle meshes obtained with a high accuracy to the voxel models [8, 9].

- Study of an EVM or CUDB-based data structure to encode time-varying datasets. A variant of the PEVE structure used by the simplification method presented in this thesis can be applied to this kind of datasets, where each 3D frame corresponds to one object. If a dataset slightly changes frame-to-frame, it can be efficiently encoded. However, in the opposite case, PEVE structure is not suitable and a new approach must be devised.

APPENDIX A

List of Publications

- I. Cruz-Matías and D. Ayala. A New Lossless Orthogonal Simplification Method for 3D Objects Based on Bounding Structures. *Graphical Models*, (Acceptable for publication provided minor revisions are made), 2013.

- I. Cruz-Matías and D. Ayala. Merging faces: A New Orthogonal Simplification of Solid Models. In R. Gonzalez-Diaz, M.-J. Jimenez, and B. Medrano, editors, *Discrete Geometry for Computer Imagery*, volume 7749 of *Lecture Notes in Computer Science*, pages 143–154. Springer-Verlag, 2013.

- I. Cruz-Matías and D. Ayala. An Efficient Alternative to Compute the Genus of Binary Volume Models. *In Proceedings of the International Conference on Computer Graphics Theory and Applications-GRAPP 2013*, pages 18–26, Barcelona, Spain, 2013.

- M. Garcia-Valles, H. Hafez, I. Cruz-Matías, E. Vergés, M. Aly, J. Nogués, D. Ayala, and S. Martínez. Calculation of viscosity-temperature curves for glass obtained from four wastewater treatment plants in Egypt. *Journal of Thermal Analysis and Calorimetry*, 111:107–114, 2013.

- D. Ayala, E. Vergés, and I. Cruz-Matías. A Polyhedral Approach to Compute the Genus of a Volume Dataset. *In Proceedings of the International Conference on Computer Graphics Theory and Applications-GRAPP 2012*, pages 38–47, Rome, Italy, 2012.

- I. Cruz-Matías and D. Ayala. Orthogonal Simplification of Objects Represented by the Extreme Vertices Model. *In Proceedings of the International Conference on Computer Graphics Theory and Applications-GRAPP 2012*, pages 193–196, Rome, Italy, 2012.

- I. Cruz-Matías and D. Ayala. CUDB: An Improved Decomposition Model for Orthogonal Pseudo-Polyhedra. Technical Report LSI-11-2-T, UPC, 2011.

- J. Rodríguez, I. Cruz, E. Vergés, and D. Ayala. A connected-component-labeling-based approach to virtual porosimetry. *Graphical Models*, 73:296–310, 2011.

- J. Rodríguez, I. Cruz, E. Vergés, and D. Ayala. Skeletonless Porosimeter Simulation. In M. Choverand and M. O. Eds., editors, *Proceedings of XX Spanish Computer Graphics Conference-CEIG 2010*, pages 49–56. Ed. Ibergarceta, 2010.

Bibliography

[1] A. Aguilera. *Orthogonal Polyhedra: Study and Application*. PhD thesis, LSI-Universitat Politècnica de Catalunya, 1998.

[2] A. Aguilera and D. Ayala. Orthogonal Polyhedra as Geometric Bounds in Constructive Solid Geometry. In *Fourth ACM Symposium on Solid Modeling and Applications*, pages 56 – 67. ACM, 1997.

[3] A. Aguilera and D. Ayala. *Geometric Modeling*, volume 14 of *Computing Supplement*, chapter Converting Orthogonal Polyhedra from Extreme Vertices Model to B-Rep and to Alternating Sum of Volumes, pages 1 – 28. Springer-Verlag, 2001.

[4] P. Alliez and M. Desbrun. Progressive compression for lossless transmission of triangle meshes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 195–202, New York, NY, USA, 2001. ACM.

[5] J. Andrade, M. Almeida, J. Mendes, J. Havlin, S. Suki, and H. Stanley. Fluid flow through porous media: The role of stagnant zones. *Physical Review Letters*, 79:3901–3904, 1997.

[6] C. Andújar. *Octree-based Simplification of Polyhedral Solids*. PhD thesis, LSI-Universitat Politècnica de Catalunya, 1999.

[7] C. Andújar, P. Brunet, and D. Ayala. Topology-reducing surface simplification using a discrete solid representation. *ACM Trans. on Graphics*, 21(2):88 – 105, 2002.

[8] C. Andújar, P. Brunet, A. Chica, I. Navazo, J. Rossignac, and A. Vinacua. Computing maximal tiles and application to impostor-based simplification. In *Computer Graphics Forum*, volume 23, pages 401–410. Wiley Online Library, 2004.

[9] C. Andujar, P. Brunet, A. Chica, I. Navazo, J. Rossignac, and A. Vinacua. Optimal iso-surfaces. *Computer-Aided Design and Applications*, 1(1-4):503–511, 2004.

[10] A. Asmara, U. Nienhuis, and R. Hekkenberg. Approximate orthogonal simplification of 3D model. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–4, july 2010.

[11] D. Ayala and E. Vergés. Improved virtual porosimeter. In *CASEIB'08*, 2008.

[12] D. Ayala and E. Vergés. Structural parameters computation of a volume using alternative representations. In J. R. F. Serón, O. Rodríguez and E. C. Eds., editors, *Proceedings of IV Iberoamerican Symposium in Computer Graphics*, pages 73–80. DJ Editores, C.A., 2009.

[13] D. Ayala, E. Vergés, and I. Cruz. A polyhedral approach to compute the genus of a volume dataset. In *Proceedings of the International Conference on Computer Graphics Theory and Applications-GRAPP 2012*, pages 38–47, Rome, Italy, February 2012. INSTICC Press.

[14] R. P. Banerjee and J. R. Rossignac. Topologically exact evaluation of polyhedra defined in CSG with loose primitives. In *Computer Graphics Forum*, volume 15, pages 205–217. Wiley Online Library, 1996.

[15] J. L. Bentley. Multidimensional binary search trees used for associative search. *Communications ACM*, 19(9):509–516, 1975.

[16] T. Biedl and B. Genç. Reconstructing orthogonal polyhedra from putative vertex sets. *Computational Geometry*, 44(8):409 – 417, 2011.

[17] A. Biswas, P. Bhowmick, M. Sarkar, and B. B. Bhattacharya. A linear-time combinatorial algorithm to find the orthogonal hull of an object on the digital plane. *Information Sciences*, 216(0):176 – 195, 2012.

[18] E. Boek and M. Venturoli. Lattice-Boltzmann studies of fluid flow in porous media with realistic rock geometries. *Computers and Mathematics with Applications*, 59:2305–2314, 2010.

[19] O. Bournez, O. Maler, and A. Pnueli. Orthogonal polyhedra: Representation and computation. *Hybrid Systems: Computation and Control*, pages 46–60, 1999. LNCS 1569, Springer.

[20] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[21] P. Brunet and M. I. Navazo. Solid representation and operation using extended octrees. *ACM Transactions on Graphics*, 9:170–197, 1990.

[22] J.-W. Chang, W. Wang, and M.-S. Kim. Efficient collision detection using a dual obb-sphere bounding volume hierarchy. *Computer-Aided Design*, 42(1):50 – 57, 2010.

[23] P. Cignoni, D. Costanza, C. Montani, C. Rocchini, and R. Scopigno. Simplification of tetrahedral meshes with accurate error evaluation. In *Proceedings of the IEEE Visualization Conference*, pages 85–92. IEEE Computer Society, 2000.

[24] P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers and Graphics*, 22(1):37–54, 1998.

[25] T. V. Cleynenbreugel. *Porous scaffolds for the replacement of large bone defects. a biomechanical design study.* PhD thesis, Katholieke Universiteit Leuven, 2005.

[26] V. Cnudde, A. Cwirzen, B. Maddchaele, and P. J. S. Jacobs. Porosity and microstructure characterization of building stones and concretes. *Engineering geology*, 103:76 – 83, 2009.

[27] J. D. Cohen, M. C. Lin, D. Manocha, and M. Ponamgi. I-collide: an interactive and exact collision detection system for large-scale environments. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, I3D '95, pages 189–ff. ACM, 1995.

[28] D. S. Coming and O. G. Staadt. Velocity-aligned discrete oriented polytopes for dynamic collision detection. *IEEE Transactions on Visualization and Computer Graphics*, 14:1–12, 2008.

[29] A. T. Corey. *Influence of shape on the fall velocity of sand grains.* Audio Visual Service, Colorado State University., 1963.

[30] T. Dang and O. Maler. Reachability analysis via face lifting. In *Hybrid Systems: Computation and Control*, pages 96–109. Springer, 1998.

[31] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry.* Springer-Verlag, 2000.

[32] J. Delerue, S. Lomov, R. Parnas, I. Verpoest, and M. Wevers. Pore network modeling of permeability for textile reinforcements. *Polymer Composites*, 24(3):244 – 357, 2003.

[33] M. Diepenbroek, A. Bartholomä, and H. Ibbeken. How round is round? a new approach to the topic 'roundness' by fourier grain shape analysis. *Sedimentology*, 39(3):411–422, 1992.

[34] M. Dillencourt, H. Samet, and M. Tamminen. A general approach to connected-component labeling for arbitrary image representations. *Journal of the ACM*, 39(2):253 – 280, 1992.

[35] G. Drevin and L. Vincent. Granulometric determination of sedimentary rock particle roundness. In *Proceedings of International Symposium on Mathematical Morphology (ISMM)*, pages 315–325, 2002.

[36] D. Eppstein and E. Mumford. Steinitz theorems for orthogonal polyhedra. In *Proceedings of the 2010 annual symposium on Computational geometry*, pages 429–438. ACM, 2010.

[37] C. Esperança and H. Samet. Orthogonal polygons as bounding structures in filter-refine query processing strategies. In *Advances in Spatial Databases*, volume 1262, pages 197–220. Springer-Verlag, 1997.

[38] C. Esperança and H. Samet. Vertex representations and their applications in computer graphics. *The Visual Computer*, 14:240–256, 1998.

[39] H. Freeman. On the encoding of arbitrary geometric configurations. *IRE Trans. Electron. Comput*, pages 260–268, 1961.

[40] R. Fuchs, V. Welker, and J. Hornegger. Non-convex polyhedral volume of interest selection. *Journal of Computerized Medical Imaging and Graphics*, 34(2):105–113, 2010.

[41] N. Gagvani and D. Silver. Shape-based volumetric collision detection. In *Proceedings of the 2000 IEEE symposium on Volume visualization*, VVS '00, pages 57–61, New York, NY, USA, 2000. ACM.

[42] P.-M. Gandoin and O. Devillers. Progressive lossless compression of arbitrary simplicial complexes. *ACM Trans. Graph.*, 21(3):372–379, July 2002.

[43] E. J. Garboczi, D. P. Bentz, and N. S. Martys. *Methods in the physics of porous media*, volume 35, chapter Digital images and computer modeling, pages 1 – 41. Academic Press, 1999.

[44] F. Geleri, O. Tosun, and H. Topcuoglu. Parallelizing broad phase collision detection algorithms for sampling based path planners. In *Proc. of the 21st Euromicro Int. Conference on Parallel, Distributed, and Network-Based Processing*, pages 384–391. IEEE, 2013.

[45] A. Glassner. *An Introduction to Ray Tracing.* Academic Press, 1989.

[46] K. M. Golden, H. Eicken, A. L. Heaton, J. Miner, D. J. Pringle, and J. Zhu1. Thermal evolution of permeability and microstructure in sea ice. *Geophysical Research Letters*, 34(34):1–6, 2007.

[47] S. Gottschalk. *Collision queries using oriented bounding boxes.* PhD thesis, The University of North Carolina, 2000.

[48] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: a hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 171–180. ACM, 1996.

[49] D. J. Graham and N. G. Midgley. Graphical representation of particle shape using triangular diagrams: an excel spreadsheet method. *Earth Surface Processes and Landforms*, 25(13):1473–1477, 2000.

[50] A. Greßand R. Klein. Efficient representation and extraction of 2-manifold isosurfaces using kd-trees. *Graphical Models*, 66:370 – 397, 2004.

[51] G. J. Grevera, J. K. Udupa, and D. Odhner. An Order of Magnitude Faster Isosurface Rendering in Software on a PC than Using Dedicated, General Purpose Rendering Hardware. *IEEE Transactions Visualization and Computer Graphics*, 6(4):335–345, 2000.

[52] E. Grisan, M. Foracchia, and A. Ruggeri. A novel method for the automatic evaluation of retinal vessel tortuosity. In *Proceedings of the 25th Annual International Conference of the IEEE EMBS, 2003.*, volume 1, pages 866 – 869 Vol.1, Cancun, Mexico, sept. 2003.

[53] S.-H. Guan, M.-K. Hsieh, C.-C. Yeh, and B.-Y. Chen. Enhanced 3D model retrieval system through characteristic views using orthogonal visual hull. In *ACM SIGGRAPH 2004 Posters*, SIGGRAPH '04, New York, NY, USA, 2004. ACM Press.

[54] D. Günther. *Topological analysis of discrete scalar data.* PhD thesis, Saarland University, 2012.

[55] N. Hagbi and J. El-Sana. Carving for topology simplification of polygonal meshes. *Comput. Aided Des.*, 42:67–75, January 2010.

[56] K. D. M. Harris, M. Tremayne, and B. M. Kariuki. Contemporary advances in the use of powder X-ray diffraction for structure determination. *Angewandte Chemie International Edition*, 40(9):1626–1651, 2001.

[57] W. Harvey, Y. Wang, and R. Wenger. A randomized $O(m \log m)$ time algorithm for computing Reeb graphs of arbitrary simplicial complexes. In *Proceedings of the 2010 annual symposium on Computational geometry*, SoCG '10, pages 267–276, New York, NY, USA, 2010. ACM.

[58] Y. Hayakawa and T. Oguchi. Evaluation of gravel sphericity and roundness based on surface-area measurement with a laser scanner. *Computers & geosciences*, 31(6):735–741, 2005.

[59] M. Hilpert and C. T. Miller. Pore-morphology-based simulation of drainage in totally wetting porous media. *Advances in water resources*, 24:243 – 255, 2001.

[60] J. Hopcroft and R. Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

[61] P. Huang and C. Wang. Volume and complexity bounded simplification of solid model represented by binary space partition. In *Proceedings - 14th ACM Symposium on Solid and Physical Modeling, SPM'10*, pages 177–182. ACM, 2010.

[62] J. Hyväluoma, P. Raiskinmäki, A. Jäsberg, A. Koponen, M. Kataja, and J. Timonen. Evaluation of a lattice-boltzmann method for mercury intrusion porosimetry simulations. *Future Generation Computer Systems*, 20:1003–1011, 2004.

[63] J. J. Jiménez, F. R. Feito, and R. J. Segura. Tetra-trees properties in graphic interaction. *Graphical Models*, 73(5):182 – 201, 2011.

[64] P. Jiménez, F. Thomas, and C. Torras. 3D collision detection: A survey. *Computers & Graphics*, 25(2):269–285, 2000.

[65] A. Kaufman. *Volume Visualization.* IEEE Computer Society Press, August 1990.

[66] A. Kaufman. *Course Annotes SIGGRAPH*, chapter Volume Visualization. 1993.

[67] A. Kaufman, D. Cohen, and R. Yagel. Volume graphics. *Computer*, pages 51–64, July 1993.

[68] M. Khachan, P. Chenin, and H. Deddi. Polyhedral representation and adjacency graph in n-dimensional digital images. *Computer Vision and Image understanding*, 79:428 – 441, 2000.

[69] B. Kim, J. Seo, and Y. Shin. Binary volume rendering using Slice-based Binary Shell. *The Visual Computer*, 17:243 – 257, 2001.

[70] D.-J. Kim, L. Guibas, and S.-Y. Shin. Fast collision detection among multiple moving spheres. *Visualization and Computer Graphics, IEEE Transactions on*, 4(3):230 –242, jul-sep 1998.

[71] J. H. Kinney, J. S. Stolken, T. S. Smith, J. T. Ryaby, and N. E. Lane. An orientation distribution function for trabecular bone. *Bone*, 36:193 – 201, 2005.

[72] J. Klosowski, M. Held, J. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *Visualization and Computer Graphics, IEEE Transactions on*, 4(1):21 –36, 1998.

[73] T. Kong and A. Rosenfeld. Digital topology: Introduction and survey. *Computer Vision, Graphics and Image Processing*, 48:357–393, 1989.

[74] S. Konkle, P. Moran, B. Hamann, and K. Joy. Fast methods for computing isosurface topology with Betti numbers. In *Data Visualization: the sate of the art proceedings Dagstuhl Seminar on Scientific Visualization*, pages 363 – 376, 2003.

[75] W. C. Krumbein. Measurement and geological significance of shape and roundness of sedimentary particles. *Journal of Sedimentary Research*, 11(2):64–72, 1941.

[76] J. Lachaud and A. Montanvert. Continuous analogs of digital boundaries: A topological approach to iso-surfaces. *Graphical Models*, 62:129 – 164, 2000.

[77] E. LaMar, B. Hamann, and K. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *IEEE Visualization'99*, pages 355–361, 1999.

[78] T. Larsson and T. Akenine-Möller. Bounding volume hierarchies of slab cut balls. *Computer Graphics Forum*, 28(8):2379–2395, 2009.

[79] L. Latecki. 3D Well-Composed Pictures. *Graphical Models and Image Processing*, 59(3):164–172, May 1997.

[80] L. Latecki, U. Eckhardt, and A. Rosenfeld. Well-composed sets. *Computer Vision and Image Understanding*, 61(1):70–83, January 1995.

[81] A. Laurentini. The visual hull concept for silhouette-based image understanding. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(2):150–162, 1994.

[82] E. Lee, Y. Choi, and K. Park. A method of 3D object reconstruction from a series of cross-sectional images. *IEICE trans. inf and syst*, E77-D(9), 1994.

[83] H. Lee, M. Desbrun, and P. Schröder. Progressive encoding of complex isosurfaces. *ACM Trans. Graph.*, 22(3):471–476, July 2003.

[84] Z. Liang, M. A. Ioannidis, and I. Chatzis. Geometric and Topological Analysis of Three-Dimensional Porous Media: Pore Space Partitioning Based on Morphological Skeletonization. *Journal of Colloid and Interface Science*, 221:13 – 24, 2000.

[85] W. B. Lindquist and A. Venkatarangan. Investigating 3D Geometry of Porous Media from High Resolution Images. *Phys. Chem. Earth (A)*, 25(7):593 – 599, 1999.

[86] A. Lopes and K. Brodlie. Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing. *IEEE Transactions on Visualization and Computer Animation*, 9(1):111–114, jan - march 2003.

[87] W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *ACM Computer Graphics*, 21(4):163–169, July 1987.

[88] N. Y. Luon, M. Z. Yusoff, and N. H. Shuaib. Tetrahedral mesh generator for CFD simulation of complex geometry. In *Energy and Environment, 2009. ICEE 2009. 3rd International Conference on*, pages 330 –336, dec. 2009.

[89] M. Mantyla. *An Introduction to Solid Modeling.* Computer Science Press, 1988.

[90] E. Martín-Badosa, A. Elmoutaouakkil, S. Nuzzo, D. Amblard, L. Vico, and F. Peyrin. A method for the automatic characterization of bone architecture in 3D mice microtomographic images. *Computerized Medical Imaging and Graphics*, 27:447–458, 2003.

[91] J. Martínez, N. Pla, and M. Vigo. Skeletal representations of orthogonal shapes. *Graphical Models*, 75:189 – 207, 2013.

[92] W. S. Massey. *A Basic Course in Algebraic Topology.* Springer-Verlag, 1991.

[93] F. J. Melero. *BP-Octree: Una Estructura Jerárquica de Volúmenes Envolventes.* PhD thesis, Universidad de Granada, 2008.

[94] F. J. Melero, P. Cano, and J. C. Torres. Bounding-planes octree: A new volume-based lod scheme. *Computers & Graphics*, 32(4):385–392, Aug. 2008.

[95] J. Menon, R. J. Marisa, and J. Zagajac. More powerful solid modeling through ray representations. *Computer Graphics and Applications, IEEE*, 14(3):22–35, 1994.

[96] J. W. Milnor. *Morse theory.* Princeton university press, 1963.

[97] J. C. Mullikin and P. W. Verbeek. Surface area estimation of digitized planes. *Bioimaging*, 1(1):6–16, 1993.

[98] M. J. Muuss and L. A. Butler. *State of the Art in Computer Graphics: Visualization and Modeling*, chapter Combinatorial solid geometry, boundary representations, and non-manifold geometry, pages 185–223. Springer-Verlag, 1991.

[99] M. Natrella. *NIST/SEMATECH e-Handbook of Statistical Methods*. NIST/SEMATECH, July 2010.

[100] B. Naylor, J. Amanatides, and W. Thibault. Merging BSP trees yields polyhedral set operations. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, volume 24, pages 115–124, 1990.

[101] T. P. Nguyen and I. Debled-Rennesson. Curvature estimation in noisy curves. In *Computer Analysis of Images and Patterns*, pages 474–481. Springer, 2007.

[102] G. Nielson. Volume modelling. In M. C. et al., editor, *Volume Graphics*, pages 29–48. Springer-Verlag, 2000.

[103] A. Odgaard. Three-dimensional methods for quantification of cancellous bone architecture. *Bone*, 20(4):315 – 328, 1997.

[104] A. Odgaard and H. J. Gundersen. Quantification of Connectivity in Cancellous Bone, with Special Emphasis on 3-D Reconstructions. *Bone*, 14:173 – 182, 1993.

[105] M. A. Otaduy and M. C. Lin. Sensation preserving simplification for haptic rendering. In *ACM SIGGRAPH 2005 Courses*, page 72. ACM, 2005.

[106] S. Ovaysi and M. Piri. Direct pore-level modeling of incompressible fluid flow in porous media. *Journal of Computational Physics*, 229:7456–7476, September 2010.

[107] R. Pajarola and J. Rossignac. SQUEEZE: Fast and progressive decompression of triangle meshes. In *Proceedings of Computer Graphics International Conference*, pages 173–182. IEEE Computer Society, 2000.

[108] S. C. Park and B. K. Choi. Boundary extraction algorithm for cutting area detection. *Computer-Aided Design*, 33(8):571–579, 2001.

[109] T. Park, H. Lee, and C.-h. Kim. Progressive compression of geometry information with smooth intermediate meshes. In *Pattern Recognition and Image Analysis*, volume 4478 of *Lecture Notes in Computer Science*, pages 89–96. Springer-Verlag, 2007.

[110] M. Pauly, M. Gross, and L. P. Kobbelt. Efficient simplification of point-sampled surfaces. In *Proceedings of the conference on Visualization '02*, VIS '02, pages 163–170, Washington, DC, USA, 2002. IEEE Computer Society.

[111] J. Peng and C.-C. J. Kuo. Geometry-guided progressive lossless 3D mesh coding with octree (ot) decomposition. *ACM Trans. Graph.*, 24(3):609–616, July 2005.

[112] R. Pérez-Aguila. Computing the discrete compactness of orthogonal pseudo-polytopes via their nD-EVM representation. *Mathematical Problems in Engineering*, 2010:Article ID 598910, 28 p., 2010.

[113] F. Peyrin, Z. Peter, A. Larrue, A. Bonnassie, and D. Attali. Local geometrical analysis of 3D porous network based on medial axis: Application to bone micro-architecture microtomography images. *Image Analysis and Stereology*, 26(3):179 – 185, 2007.

[114] P. Pivonka and S. V. Komarova. Mathematical modeling in bone biology: From intracellular signaling to tissue mechanics. *Bone*, 47(2):181 – 189, 2010.

[115] L. Pothuaud, D. C. Newitt, Y. Lu, B. MacDonald, and S. Majumdar. In vivo application of 3D-line skeleton graph analysis (LSGA) technique with high-resolution magnetic resonance imaging of trabecular bone structure. *Osteoporos Int.*, 15:411 – 419, 2004.

[116] L. Pothuaud, B. V. Rietbargen, L. Mosekilde, O. Beuf, P. Levitz, and C. L. Benhamou. Combination of topological parameters and bone volume fraction better predicts the mechanical properties of trabecular bone. *Journal of Biomechanics*, 35:1091 – 1099, 2002.

[117] D. Pountain. Run-length encoding. *Byte*, 12(6):317–319, 1987.

[118] W. R. Quadros, K. Shimada, and S. J. Owen. 3D discrete skeleton generation by wave propagation on PR-octree for finite element mesh sizing. In *Proceedings of ACM Symposium on Solid Modeling and Applications*, pages 327 – 332, 2004.

[119] A. Rappoport. The n-dimensional extended convex differences tree (ECDT) for representing polyhedra. In *Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications*, SMA '91, pages 139–147, New York, NY, USA, 1991. ACM.

[120] A. Rappoport. An efficient adaptive algorithm for constructing the convex differences tree of a simple polygon. In *Computer graphics forum*, volume 11, pages 235–240. Wiley Online Library, 1992.

[121] A. Requicha. Representations for rigid solids: Theory, methods and systems. *ACM Computing Surveys*, 12(4):73–82, Dec. 1980.

[122] L. Rico, A. Naranjo, S. Noriega, E. Martínez, and L. Vidal. Effect of cutting parameters on the roundness of cylindrical bars turned of 1018 steel. *Measurements*, 2010.

[123] O. Ripolles, M. Chover, J. Gumbau, F. Ramos, and A. Puig-Centelles. Rendering continuous level-of-detail meshes by masking strips. *Graphical Models*, 71(5):184–195, 2009.

[124] J. Rodríguez and D. Ayala. Fast neighborhood operations for images and volume data sets. *Computers & Graphics*, 27:931–942, 2003.

[125] J. Rodríguez, D. Ayala, and A. Aguilera. *Geometric Modeling for Scientific Visualization*, chapter EVM: A Complete Solid Model for Surface Rendering, pages 259–274. Springer-Verlag, 2004.

[126] J. Rodríguez, I. Cruz, E. Vergés, and D. Ayala. A connected-component-labeling-based approach to virtual porosimetry. *Graphical Models*, 73:296–310, 2011.

[127] J. E. Rodríguez. *Contribution to Surface/Volume Integration: A Model for Visualization and Manipulation.* PhD thesis, LSI-UPC, 2004.

[128] A. Rosenfeld and J. Pfaltz. Sequential operations in digital picture processing. *Journal of the ACM*, 13(4):471–494, 1966.

[129] J. Rossignac. CSG formulations for identifying and for trimming faces of CSG models. In *CSG'96*, volume 96, pages 1–14, 1996.

[130] J. R. Rossignac and H. B. Voelcker. Active zones in CSG for accelerating boundary evaluation, redundancy elimination, interference detection, and shading algorithms. *ACM Transactions on Graphics (TOG)*, 8(1):51–87, 1988.

[131] T. Roussillon, H. Piégay, I. Sivignon, L. Tougne, and F. Lavigne. Automatic computation of pebble roundness using digital imagery and discrete geometry. *Computers & Geosciences*, 35(10):1992–2000, 2009.

[132] K. I. Rybakov, V. Semenov, G. Link, and M. Thumm. Preferred orientation of pores in ceramics under heating by a linearly polarized microwave field. *Journal of Applied Physics*, 101(8):084915–084915–5, 2007.

[133] N. Sahu and S. Panigrahi. Mathematical aspects of rietveld refinement and crystal structure studies on pbtio3 ceramics. *Bulletin of Materials Science*, 34(7):1495–1500, 2011.

[134] H. Samet. *Applications of spatial data structures: Computer graphics, image processing, and GIS.* Addison-Wesley Longman Publishing Co., Inc., 1990.

[135] H. Samet and A. Kochut. Octree approximation and compression methods. In *Proceedings of First International Symposium on 3D Data Processing Visualization and Transmission*, pages 460–469. IEEE Computer Society, 2002.

[136] H. Samet and M. Tamminen. Bintrees, CSG trees, and time. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '85, pages 121–130, New York, NY, USA, 1985. ACM.

[137] H. Samet and R. Webber. Hierarchical data structures and algorithms for computer graphics. *IEEE Computer Graphics & Applications*, 8(3):48–68, May 1988.

[138] H. Sánchez-Cruz, E. Bribiesca, and R. M. Rodríguez-Dagnino. Efficiency of chain codes to represent binary objects. *Pattern Recognition*, 40:1660 – 1674, 2007.

[139] V. V. Savchenko, A. A. Pasko, A. Sourin, and T. L. Kunii. Volume modelling: Representations and advanced operations. In *Computer Graphics International*, pages 4–13, 1998.

[140] S. Schaefer, T. Ju, and J. Warren. Manifold dual contouring. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):610 – 619, 2007.

[141] G. Schena and S. Favretto. Pore space network characterization with sub-voxel definition. *Transp. Porous Media*, 70:181 – 190, 2007.

[142] H. Scholze. Influence of viscosity and surface tension on hot stage microscopy measurements on glasses. *Ber. Dtsch. Keram. Ges.*, 391:63 – 68, 1962.

[143] M. Schroth, J. Istok, S. Ahearn, and J. Selker. Characterization of miller-similar silica sands for laboratory hydrologic studies. *Soil Science Society of America Journal*, 60(5):1331–1339, 1996.

[144] W. Schulz, J. Becker, A. Wiegmann, P. P. Mukherjee, and C. Wang. Modeling of two-phase behavior in the gas diffusion medium of PEFCs via full morphology approach. *Journal of The Electrochemical Society*, 154(4):B419–B426, 2007.

[145] T. Seidl, B. Boden, and S. Fries. Cc-mr–finding connected components in huge graphs with mapreduce. In *Machine Learning and Knowledge Discovery in Databases*, pages 458–473. Springer, 2012.

[146] P. Shirley, M. Ashikhmin, M. Gleicher, S. Marschner, E. Reinhard, K. Sung, W. Thompson, and P. Willemsen. *Fundamentals of computer graphics.* AK Peters Natick, 2002.

[147] D. B. Silin, G. Jin, and T. W. Patzek. Robust determination of the pore space morphology in sedimentary rocks. *Journal of Petroleum Technology*, pages 69 – 70, 2004.

[148] E. D. Sneed and R. L. Folk. Pebbles in the lower colorado river, texas a study in particle morphogenesis. *The Journal of Geology*, pages 114–150, 1958.

[149] M. Stauber and R. Müller. Volumetric spatial decomposition of trabecular bone into rods and plates: A new method for local bone morphometry. *Bone*, 38(4):475–484, 2006.

[150] P. Stroeven and Z. Guo. Modern routes to explore concrete's complex pore space. *Image Anal Stereol*, 25(2):75–86, 2006.

[151] R. Sun, S. Gao, and W. Zhao. An approach to b-rep model simplification based on region suppression. *Computers &amp; Graphics*, 34(5):556 – 564, 2010.

[152] W. Sun, A. Darling, B. Starly, and J. Nam. Computer-aided tissue engineering: overview, scope and challenges. *Biotechnol. Appl. Biochem.*, 39:29 – 47, 2004.

[153] W. Sun, B. Starly, J. Nam, and A. Darling. Bio-CAD modeling and its applications in computer-aided tissue engineering. *Computer-Aided Design*, 37(11):1097–1114, 2005.

[154] S. Suri, P. M. Hubbard, and J. F. Hughes. Analyzing bounding boxes for object intersection. *ACM Trans. Graph.*, 18:257–277, July 1999.

[155] M. Tarini, N. Pietroni, P. Cignoni, D. Panozzo, and E. Puppo. Practical quad mesh simplification. *Computer Graphics Forum (Special Issue of Eurographics 2010 Conference)*, 29(2):407–418, 2010.

[156] Y. Tashiro. On methods for generating uniform random points on the surface of a sphere. *Annals of the Institute of Statistical Mathematics*, 29(1):295–300, 1977.

[157] T. Theile and M. Schneebeli. Algorithm to decompose three-dimensional complex structures at the necks: tested on snow structures. *Image Processing, IET*, 5(2):132–140, 2011.

[158] W. Thibault and B. Naylor. Set operations on polyhedra using binary space partitioning trees. *SIGGRAPH Comput. Graph.*, 21(4):153–162, 1987.

[159] L. Thurfjell, E. Bengtsson, and B. Nordin. A boundary approach to fast neighborhood operations on three-dimensional binary data. *CVGIP: Graphical Models and Image Processing*, 57(1):13 – 19, 1995.

[160] R. Tilove and A. Requicha. Closure of boolean operations on geometric entities. *Computer-Aided Design*, 12(5):219 – 220, 1980.

[161] J. Toriwaki and T. Yonekura. Euler number and connectivity indexes of a three dimensional digital picture. *Forma*, 17:183–209, 2002.

[162] I. J. Trotts, B. Hamann, and K. I. Joy. Simplification of tetrahedral meshes with error bounds. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):224–237, 1999.

[163] H. S. Tuan and D. W. Hutmacher. Application of micro CT and computation modeling in bone tissue engineering. *Computer-Aided Design*, 37(11):1151–1161, 2005.

[164] J. K. Udupa and D. Odhner. Fast visualization, manipulation, and analysis of binary volumetric objects. *IEEE Computer Graphics & Applications*, pages 53–62, Nov. 1991.

[165] J. K. Udupa and D. Odhner. Shell rendering. *IEEE Computer Graphics & Applications*, pages 58–67, Nov. 1993.

[166] D. Ushizima, D. Morozov, G. Weber, A. Bianchi, J. Sethian, and E. Bethel. Augmented topological descriptors of pore networks for material science. *Visualization and Computer Graphics, IEEE Transactions on*, 18(12):2041–2050, 2012.

[167] J. Vanderhyde and A. Szymczak. Topological simplification of isosurfaces in volume data using octrees. *Graphical Models*, 70:16 – 31, 2008.

[168] E. Vergés. *Modeling, Analysis and Visualization of Porous Biomaterials*. PhD thesis, LSI-Universitat Politècnica de Catalunya, 2011.

[169] E. Vergés, D. Ayala, S. Grau, and D. Tost. 3D reconstruction and quantification of porous structures. *Computers & Graphics*, 1(32):438–444, 2008.

[170] E. Vergés, D. Ayala, S. Grau, and D. Tost. Virtual porosimeter. *Computer-Aided Design and Applications*, 5(1-4):557–564, 2008.

[171] E. Vergés, D. Tost, D. Ayala, E. Ramos, and S. Grau. 3D pore analysis of sedimentary rocks. *Sedimentary Geology*, 234(1–4):109–115, 2011.

[172] M. Vigo, N. Pla, D. Ayala, and J. Martínez. Efficient algorithms for boundary extraction of 2D and 3D orthogonal pseudomanifolds. *Graphical Models*, 74:61–74, 2012.

[173] I. M. Vitez, A. W. Newman, M. Davidovich, and C. Kiesnowski. The evolution of hot-stage microscopy to aid solid-state characterizations of pharmaceutical solids. *Thermochimica Acta*, 324(1-2):187 – 196, 1998.

[174] H. J. Vogel, J. Tölke, V. Schulz, M. Krafczyk, and K. Roth. Comparison of a lattice-boltzmann model, a full-morphology model, and a pore network model for determining capillary pressure-saturation relationships. *Vadose Zone Journal*, 4:380 –388, 2005.

[175] A. Vogiannou, K. Moustakas, D. Tzovaras, and M. G. Strintzis. Enhancing bounding volumes using support plane mappings for collision detection. *Computer Graphics Forum*, 29(5):1595–1604, 2010.

[176] H. Wadell. Volume, shape, and roundness of rock particles. *The Journal of Geology*, 40:443–451, 1932.

[177] H. Wadell. Sphericity and roundness of rock particles. *The Journal of Geology*, 41(3):310–331, 1933.

[178] I. Wald, S. Boulos, and P. Shirley. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Transactions on Graphics*, 26(1), January 2007.

[179] J. Wilhems and A. V. Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.

[180] J. Williams and J. Rossignac. Tightening: Morphological simplification. *International Journal of Computational Geometry & Applications*, 17(5):487–503, 2007.

[181] G. Windreich, N. Kiryati, and G. Lohmann. Voxel-based surface area estimation: from theory to practice. *Pattern Recognition*, 36(11):2531–2541, 2003.

[182] W. Wong, F. Y. Shih, and T. Su. Thinning algorithms based on quadtree and octree representations. *Information Sciences*, 176:1379 – 1394, 2006.

[183] S.-E. Yoon, B. Salomon, M. Lin, and D. Manocha. Fast collision detection between massive models using dynamic simplification. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 136–146. ACM, 2004.

[184] G. Zachmann. Minimal hierarchical collision detection. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '02, pages 121–128, New York, NY, USA, 2002. ACM.