Decomposition of a 3D discrete object surface into discrete plane pieces

Isabelle Sivignon[*], Florent Dupont[†] and Jean-Marc Chassery[‡]

[*][‡]Laboratoire LIS
961, rue de la Houille Blanche
Domaine Universitaire - BP46
38402 Saint Martin D'Hères Cedex, France

[†]Laboratoire LIRIS
8, Boulevard Niels Bohr
69622 Villeurbanne Cedex, France

Abstract

This paper deals with the polyhedrization of discrete volumes. The aim is to do a reversible transformation from a discrete volume to a Euclidean polyhedron, i.e. such that the discretization of the Euclidean volume is exactly the initial discrete volume. We propose a new polynomial algorithm to split the surface of any discrete volume into pieces of naive discrete planes with well-defined shape properties, and present a study of the time complexity as well as a study of the influence of the voxel tracking order during the execution of this algorithm.

Keywords: discrete volumes, digital plane recognition, surface, polyhedrization.

1 Introduction

3D discrete volumes are more and more used especially in the medical area since they result from MRI and scanners. As 2D images are composed of squares called pixels, these 3D images are composed of cubes called voxels. This structure induces many difficulties in the exploitation and study of these objects: as each cube is stored, the volume of data is very huge, which is a problem to get a fluent interactive visualization; the facet structure (voxels's faces) of the discrete object induces many problems to get a nice visualization that is necessary for medicines, as no rendering nor texture algorithm can be applied.

The general idea to solve those problems is to transform discrete volumes into Euclidean polyhedra. Many research activities have already been

[*]Corresponding author: sivignon@lis.inpg.fr, fax number: 0033476826256
[†]fdupont@ligim.univ-lyon1.fr
[‡]chassery@lis.inpg.fr

achieved to find solutions to this problem, using Euclidean geometry or discrete geometry. To get a good visualization of discrete volumes, the method that is most used is the Marching cubes method [1], which considers local voxel configurations to replace them by small triangles. Even if this method offers a good visualization, it does not provide a good data compression (huge number of facets) and is not reversible.

Many other research activities have been done in this field, using completely different ideas. The first algorithms dealt with the construction of the convex hull of the considered set of voxels. This study was mainly done by Kim and Rosenfeld who published in [2] a first algorithm to characterize a piece of discrete plane by the convex hull of the discrete surface. This algorithm was then improved by Kim and Stojmenović [3]. This algorithm was not reversible, i.e. the discretization of the Euclidean hull obtained is not the discrete object.

The first reversible algorithm was proposed by Borianne and Françon [4]. In this paper, they expose two methods: one to do a polyhedrization, and another to do the reverse operation, i.e. discretization. For that, they use an approximation by the least-square method that make it marginal compared with entirely discrete methods.

Another idea was then proposed by Debled [5] [6]. She developed an algorithm to recognize rectangular pieces of naive planes. Then, she uses this algorithm in order to decompose the digital surface of symmetric objects (with known symmetries) into pieces of discrete planes. The polyhedrization was not complete here but it was the first approach using discrete plane recognition.

In 1999, Papier [7] [8] presents an algorithm using the Fourier-Motskin algorithm to recognize standard discrete planes on an object surface, each point of the plane being a pointel (vertex of a voxel). The time complexity of this algorithm is high because of the Fourier-Motskin algorithm and moreover, the polyhedrization done is not reversible.

Finally, in 2000, Burguet and Malgouyres published [9] an approximation algorithm using a curvature computation to choose some germ points and then calculate the skeleton of the discrete surface without those germs (Voronoi diagram). The result is a Delaunay triangulation that approximates and simplifies the original object.

The aim of this paper is to present the first steps to achieve a totally discrete and reversible polyhedrization. We use discrete geometry that seems to fit best the structure of the processed objects. Reversibility means that from a discrete object, we can get a Euclidean polyhedron which digitalization is exactly the former discrete volume. This property enables many applications and we give two of them here. First, this can lead to an efficient data compression describing the volume by the set of all the faces of the Euclidean polyhedron: no loss of data and no loss of information in the compressed object. After this transformation, we can apply morphological operations on the reconstructed Euclidean polyhedron and then retrieve the discrete volume obtained after these operations.

In a first part, we give the basic definitions of discrete geometry. Then, we present in detail the naive plane recognition algorithm that we use in the following, giving some improvements and new properties. In section 4, after a short state of the art, we expose our splitting algorithm. Section 5 deals with the algorithm time complexity. In the next section, we propose a study of the voxel processing order and its influence on the final surface decomposition. Before a few words of conclusion, we finally present some performance and image results on generated and real volumes.

2 Basic Definitions and properties

In this first part, we focus in a few words on the basic objects definitions of discrete geometry. All the following definitions lie in a discrete 3D space. This space is defined as a unit cubic mesh centered on points having integer coordinates. The vertices of each cell (cube) of the mesh correspond to points with half-integer coordinates.

A **voxel** or $\mathbb{Z}^3$ point or discrete point is assimilated with the unit closed cubes of the mesh. Then, voxel coordinates are the coordinates of the corresponding cube center. Faces, edges and vertices of a voxel are respectively called **surfels**, **linels** and **pointels**.

In $\mathbb{Z}^3$, three voxel neighborhoods (figure 2) are classically used. They are defined with the two distances called Manhattan distance, denoted d_6 and Chess board distance, denoted d_{26}:

$$d_6(M, P) = |x_m - x_p| + |y_m - y_p| + |z_m - z_p|$$

$$d_{26}(M, P) = max(|x_m - x_p|, |y_m - y_p|, |z_m - z_p|)$$

Two voxels M and P are **6-neighbors** (6-N) if and only if $d_6(M, P) \leq 1$. M and P are **26-neighbors** (26-N) if and only if $d_{26}(M, P) \leq 1$. In other words, two points are 6-N if they have a common face, 26-N if they have a common face, a common edge or a common vertex. This point of view suggests another neighborhood for the case of two voxels sharing a common face or a common edge, called 18-N ($d_6(M, P) \leq 2$).

A classical way to define a discrete line or a discrete plane is to consider the digitization of a Euclidean line or plane on a unit grid with a given digitization scheme. But, as in Euclidean space, there exists arithmetical definitions of discrete planes and lines. Those definitions where given by Reveillès [10] and then generalized to hyperplanes by Andrès [11].

A **digital plane** (figure 3) of normal vector (a, b, c), translation parameter r and arithmetical thickness $\omega \in \mathbb{N}$ is defined as the set of points

$M(x, y, z) \in \mathbb{Z}^3$ satisfying the double inequality:

$$0 \leq ax + by + cz + r < \omega$$

where a, b, c are not all null and satisfy $gcd(a, b, c) = 1$. A discrete plane such that $\omega = |a| + |b| + |c|$ is called **standard**.
A discrete plane such that $\omega = max(|a|, |b|, |c|)$ is called **naive**. (see figure 3 for an example)

The thickness parameter determines the connectivity of the plane. In fact, naive planes are the thinnest connected planes without holes and therefore they are very well adapted for object surface study. In the rest of the paper, we will deal with naive planes denoted $P(a, b, c, r)$.

Finally, naive discrete plane can be decomposed into primitive elements called **tricubes**: the **tricube** at point (i, j) of the naive plane P is defined as the set $\{(x, y, z) \in P \mid i \leq x \leq i + 3,\ j \leq x \leq j + 3\}$.

3 Recognition of a piece of discrete naive plane

We present in this part an algorithm proposed by Vittone and Chassery [12] to recognize digital plane segments. Some new properties are also proved.

3.1 Description of the algorithm

Given a Euclidean plane P defined by $ax + by + cz + r = 0$, where $0 \leq a \leq b \leq c$ and $c \neq 0$, the *OBQ discretization*(*Object Boundary Quantization*) of P is the set of all points $M(x, y, z)$ of the mesh on or "under" P. For $x, y \in \mathbb{Z}$, this method consists of rounding z to the lower integer value. The result of such a discretization is the naive plane with parameters (a, b, c, r).

In [13, 12], Vittone presents an algorithm that solves in polynomial time the following problem (so called recognition problem):
Let S be a set of voxels containing the origin $(0, 0, 0)$ and n other voxels (i_q, j_q, k_q), $q = 1, \ldots n$. What is the set $\bar{S}$ of the parameters $(\alpha, \beta, \gamma) \in \mathbb{R}^3$ with $0 \leq \alpha \leq \beta < 1$ and $0 \leq \gamma \leq 1$ such that all the voxels of S belong to the OBQ discretization of $P : \alpha x + \beta y + z + \gamma = 0$? Then, we look for the set $\bar{S}$ defined by:

$$\bar{S} = \{(\alpha, \beta, \gamma) \in [0, 1[^2 \times [0, 1],\ \alpha \leq \beta \mid \forall (x, y, z) \in S\ \ 0 \leq \alpha x + \beta y + z + \gamma < 1\}$$

Let us consider the duality of the double inequality of the former formula. Indeed, let P be a Euclidean plane defined by $z = -(\alpha x + \beta y + \gamma)$. This equation represents all the points (x, y, z) belonging to P. Let us rewrite the equation as $\gamma = -(x\alpha + y\beta + z)$. Then, in the dual space $(0, \alpha, \beta, \gamma)$ (also called parameter space), this equation represents all the planes containing the point (x, y, z). In this space, a plane (a, b, c, r) is the point $(\frac{a}{c}, \frac{b}{c}, \frac{r}{c})$ if $c = \max(a, b, c)$.

Since each voxel generates a double inequality, in the dual space each voxel of S is represented by an half-open strip delimited by two parallel planes. For a given voxel (x, y, z), this area represents the set of Euclidean planes parameters whose OBQ discretization contains the voxel (x, y, z). Finally, $\bar{S}$ is the intersection in the dual space of n half-opened strips delimited by two Euclidean planes $P(i_q, j_q, k_q)$ and $P(i_q, j_q, k_q - 1)$, $q = 1, \ldots n$.

This is the main point of the recognition algorithm: each voxel constraints the solution area in the dual space with an half-opened strip. The intersection of those half-spaces can be found step by step adding one voxel after the other. At the end, $\bar{S}$ can be a polyhedron, a polygon, a line segment or empty. In the last case, the voxels are not coplanar.

We present here a sketch of the final algorithm. Let $M(x, y, z)$ be a voxel and S the set containing M and p other voxels with coordinates $(x + i_q, y + j_q, z + k_q)$, $q = 1, \ldots p$. The aim is to find out the set of the naive planes containing all the $p+1$ voxels of S, M being the origin. The computation of the half-spaces intersection returns the solution area $\bar{S}$ and the final solutions are, after translation, the planes $P(a, b, c, r-(ax+by+cz))$ such that $(\frac{a}{c}, \frac{b}{c}, \frac{r}{c})$ is in $\bar{S}$.

Since $0 \leq \alpha \leq \beta < 1$ and $0 \leq \gamma \leq 1$, the initial solution area is delimited by the projections of the six vertices of

$$B_0 = \{(0,0,0,1), (0,1,0,1), (1,1,0,1), (0,0,1,1), (0,1,1,1), (1,1,1,1)\}$$

(figure 4) onto the dual space. In the rest of this paper, B_q will stand for the set of the points in $\mathbb{N}^4$ such that their projections in the parameter space are the vertices of the solution area for the first q voxels. Hence, $\bar{S}$ is the projection of translated B_{p+1} in the parameter space.

Let us denote $L_q(a, b, c, r) = ai_q + bj_q + ck_q + r$ and $L_q^+(a, b, c, r) = L_q(a, b, c, r) - c$. Let (a, b, c, r) be the normal vector of a plane P solution after step q. Then, at step $q + 1$, this plane is still a solution if and only if $L_{q+1}(a, b, c, r)$ and $L_{q+1}^+(a, b, c, r)$ have opposite signs, *i.e.* in the dual space, the point corresponding to the plane P is between the two planes defined by the voxel $(i_{q+1}, j_{q+1}, k_{q+1})$.

The following algorithm takes as input a voxel $V(i_q, j_q, k_q)$ and the set B_{q-1} solution for the first $q - 1$ voxels, and computes the set B_q of the solution polyhedron vertices after the addition of V.

Function Add_voxel(B_{q-1},V)

Initialization. $B_q = \emptyset$.
$L_q(a, b, c, r) = ai_q + bj_q + ck_q + r$ and $L_q^+(a, b, c, r) = L_q(a, b, c, r) - c$.

Main loop.
(1) **For all** V_1 belonging to B_{q-1} do

(2) **If** $L_q(V_1) = 0$ or $L_q^+(V_1) = 0$ **then** put V_1 in B_q
(3) **Else if** $L_q(V_1) > 0$ and $L_q^+(V_1) < 0$ **then** put V_1 in B_q
(4) **Else**
(5) **For** all V_2 $in B_{q-1}$, $V_2 \neq V_1$ such that $L_q(V_1)$ and $L_q(V_2)$ or $L_q^+(V_1)$ and $L_q^+(V_2)$ have opposite signs
(6) • Compute the intersection I of the line (V_1V_2) and the plane $L_q(X) = 0$ (or $L_q^+(X) = 0$)
(7) • Put I in B_q
(8) **end for**
(9) **end for**

Result. Return B_q.

The result of this function is the set of the solution polyhedron vertices after the processing of the q first voxels. Hence, to check if a set of voxels S are coplanar, it is enough to call the function **Add_voxel** for one voxel after the other using each time the last B_q computed. In the rest of this paper, we call *recognition algorithm* the algorithm that recognizes a piece of plane.

3.2 Properties and improvements

This polyhedron $\bar{S}$ is the intersection of half open strips. Hence, although the points that are linearly dependent with positive weights to the vertices of $\bar{S}$ are necessarily solutions, this algorithm does not precise whether the vertices, edges and faces of $\bar{S}$ are solutions or not.

Proposition 1 *Let $S = \{(i_q, j_q, k_q), q = 1, \ldots, p\}$ be a set of p voxels, and let $\bar{S}$ be the solution polyhedron obtained with the recognition algorithm. If $\bar{S}$ is not empty, let $N = \{N_i, i = 1 \ldots m\}$ be the set of the vertices of $\bar{S}$. Then, N_i is a solution if and only if $\forall q, 1 \leq q \leq p, L_q^+(N_i) \neq 0$.*

Let E be a point of the edge (N_i, N_j). If N_i or N_j is a solution, then E is also a solution.

Proof: Let $N_i(a, b, c, r)$ be a vertex of $\bar{S}$. Suppose that there exists a voxel (i_q, j_q, k_q) such that $L_q^+(N_i) = 0$. This means that N_i belongs to the plane $(i_q, j_q, k_q - 1)$ in the dual space. Since this plane is the open limit of the solution area, N_i is not a solution. On the other hand, suppose that N_i is not a solution, and show that there exists a voxel (i_q, j_q, k_q) such that $L_q^+(N_i) = 0$. By construction, two kinds of non-solution points exist: those that are not in the solution polyhedron, and those that belong to an open side of the polyhedron. As N_i is a non-solution vertex of the solution polyhedron, it belongs to a plane that is an open side of the polyhedron, i.e. a plane whose normal vector is $(i_q, j_q, k_q - 1)$. Then, there exists (i_q, j_q, k_q) such that $ai_q + bj_q + c(k_q - 1) + r = 0$, and then $L_q^+(N_i) = 0$.

Let E be a point of the edge (N_i, N_j) with N_i solution. Suppose that E is not a solution. Then, there exists a half open strip that does not contain E. As E is on an edge of the polyhedron, E belongs to the open plane of a strip. Either this plane contains the edge (N_i, N_j) and then this leads to a contradiction, or this plane cuts this edge in E, and then, one of the two vertices N_i or N_j is outside the strip. If N_i is outside, then we get the contradiction. Otherwise, if N_i is solution, then N_j is not. As E is on the edge (N_i, N_j), N_j does not belong to the open plane, which implies that N_j is not a vertex of $\bar{S}$. Contradiction. □

Corollary 1 *Let E be a point of a face F of $\bar{S}$. Let $N_i, i = 1, \ldots n, n >= 2$ the set of vertices of F. If at least one N_i is a solution and if E is not on an edge of the face, then E is also a solution.*

Proof: For $n = 2$, see proposition 1. For $n > 2$, the demonstration is nearly the same. Suppose that E is not a solution. As E is on a face of the polyhedron, E belongs to one of the open planes of the strips. If this plane contains the face F, then we get the contradiction as N_i belongs to this face. Otherwise, there exists an open plane containing E. As E is not on an edge and as $\bar{S}$ is convex, this plane cuts the face F in at least two edge points. This plane splits the space into two half-spaces, one containing points that do not belong to $\bar{S}$. Therefore, at least one vertex of F will be in this half-space, contradiction. □

Now let us focus on the line (6) of the function **Add_voxel** presented in section 3.1. Many efficient algorithms exist to compute the intersection of a polyhedron and a plane (see for instance [14], chap.7). Those algorithms return the set of vertices of the polyhedron as rational numbers. But to get the plane normal vectors corresponding to the vertices coordinates, we must have those coordinates in fractional form. Instead of computing the polyhedron first and then transforming each vertex coordinate, it is better to compute them directly as fractions.

In [13], that was done using a modified version of Grabiner algorithm [15]. This algorithm uses Farey series and their properties to compute the new vertices v with a dichotomy method. The time complexity is then $\mathcal{O}(log(n))$ if v is between two vertices v_1 and v_2 such that $d(v_1, v_2) = n$ where d denote the Euclidean distance. We propose here to compute directly those coordinates keeping at each step of the computation the value of numerators and denominators. This step can be done in $\mathcal{O}(1)$ time with the following algorithm.

V_1 and V_2 are two vertices of the current solution polyhedron and P is a plane in the dual space. This algorithm will compute the parameters of the Euclidean plane whose representation in the dual space is the intersection point between the line (V_1, V_2) and the plane P.

function Plane_line(V_1,V_2,P)

Initialization. $V_1(a_1, b_1, c_1, r_1)$, $V_2(a_2, b_2, c_2, r_2)$, $P : \alpha i + \beta j + k + \gamma = 0$, in the dual space $(0, \alpha, \beta, \gamma)$.
Let p be the intersection point of the line (V_1, V_2) and the plane P.

Computation.
Compute $N = -ia_1c_2 - jb_1c_2 - r_1c_2 - kc_1c_2$.
Compute $D = i(a_2c_1 - a_1c_2) + j(b_2c_1 - b_1c_2) + (r_2c_1 - r_1c_2)$.

Result. The three coordinates have a common denominator: $p_d = N \times c_1c_2$. The three numerators are $p_n = (N(a_2c_1 - a_1c_2) + a_1c_2D,\ N(b_2c_1 - b_1c_2) + b_1c_2D,\ N(r_2c_1 - r_1c_2) + r_1c_2D)$.

It is easy to retrieve the coordinates of the corresponding plane with the definition of the dual space: for instance, if $|c| = max(|a|, |b|, |c|)$, the plane coordinates are $(N(a_2c_1 - a_1c_2) + a_1c_2D,\ N(b_2c_1 - b_1c_2) + b_1c_2D,\ p_d,\ N(r_2c_1 - r_1c_2) + r_1c_2D)$.

To conclude on this part, this recognition algorithm offers some properties that are useful for the next step, i.e. applying this algorithm on a discrete surface:

- it recognizes naive discrete plane: the minimal thickness of these planes implies that the object surface is enough to do a recognition, we do not need interior voxels;
- it is incremental: the voxels can be added one by one;
- for a given set of voxels, the adding order does not have an influence on the final result;
- it returns the set of vertices of the solution polyhedron: so, we have the complete set of the solution planes normal vectors.

4 General algorithm

Recognizing discrete planes is the first step of a most general goal: the polyhedrization of a discrete object. This section describes a new algorithm that split the discrete surface of an object into naive plane pieces. We will also see that this algorithm has features which make it especially well adapted to get a totally discrete and reversible polyhedrization.

We consider 26-connected objects with a 6-connected background. In the rest of the paper, we call *surface* the set of the surfels that belong simultaneously to an object voxel and to a background voxel. In other words, the surface is the set of visible surfels. As each voxel has six faces, those six faces define six directions that we consider symmetrically during the algorithm description.

Algorithm Decompose-discrete-surface

Initialization. For each object voxel, locate the surface surfels, S.
Initialize the number of planes *cpt* to -1.
Initialize the list `To-process` with the empty list.
Let B be a set of vertices of a solution polyhedron: B_0, the initial set, depends on the current direction.

Main loop.

(1) **For** each object direction d
(2)   **For** each object voxel V
(3)   Let s_0 be the surfel of V in the direction d;
(4)   **If** $s_0 \in S$ and s_0 has never been treated **then**
(5)   origin $= s_0$;
(6)   *cpt* = *cpt* $+ 1$;
(7)   put s_0 in `To-process`;
(8) $B = B_0$;
(9)   **While** `To-process` is not empty
(10)   choose one surfel s in `To-process`;
(11) $B_{save} = B$;
(12)   **For** each of the 8 neighbors s_n of s
(13) $B =$ **Add_voxel(**B,s_n**)**
(14)   **if** B is not empty **then**
(15)   *cpt* is a solution for s and its 8 neighbors;
(16)   among the 8 neighbors, put those which have not been treated yet for this plane into the list `To-process`;
(17)   **else**
(18)   **If** $s = s_0$ **then** *cpt* = *cpt* $- 1$ and clear `To-process`;
(19) $B = B_{save}$;
          end while
    end for
end for

Result. For each surfel: a list of all the plane numbers it belongs to.
For each piece of plane: the set of all the solution polyhedron vertices.

In this algorithm, the solution polyhedron is represented by the set of its vertices denoted by B. Each time the function **Add_voxel** is called, the set

B is modified. We save the value of B before the addition of the 8 neighbors of a given surfel s. So, if s is not a tricube center, we can recover the solution polyhedron as it was before the processing of s's neighbors (lines (18) and (19)).

During the execution, for each surfel we create a list containing all the plane numbers to which this surfel belongs. Moreover, at the end of each piece of plane recognition, we keep in an appropriate structure the coordinates of the solution polyhedron vertices.

Let us analyze the properties of this algorithm:

- during the processing of a surfel, either 8 faces are added to the current plane or zero: indeed, if a surfel is a tricube center, then we add all of them to the current plane, otherwise, none of them are added (even those which could belong to the plane). This implies that every surfel of a recognized naive plane has a least 3 neighbors belonging to this plane. Indeed, a face that belongs to a piece of plane must have a neighbor that is a tricube center. Hence, only two cases are possible (see figure 5). As a consequence, recognized regions have a "regular form";

- a surfel can belong to many pieces of planes: indeed, no restrictions nor choices are done during the expansion of the planes. Then, naive planes are extended to their maximum under the constraint given before.

The second property can be seen as an advantage or as a problem. Indeed, if we do not allow discrete plane covering, the limit between two planes is easy to handle. But we can get many very small pieces of plane at the end of the algorithm and hence, allowing plane covering reduces the influence of the seeds used for the pieces of planes. Moreover, to get a reversible polyhedrization, the border of a piece of plane should be a discrete line. Without covering, we have no mean to control the border of the pieces of plane.

5 Time Complexity

In this section, we give a polynomial bound on the algorithm time complexity. This study is split into two parts: first, the time complexity of the function **Add_voxel** presented in section 3; then, the time complexity of the algorithm **Decompose-discrete-surface** described in section 4.

5.1 Add_voxel time complexity

The first loop of this algorithm covers the elements of the set B_q. To bound the cardinality of this set, we have to bound the number of vertices of a polyhedron according to its number of faces. This is a classical result in computational geometry (see [14], chap.7 for instance) that we recall here:

Theorem 1 *Let P be a convex polyhedron with n faces. Then P has at most $2n-4$ vertices.*

In the algorithm, B_0 is a polyhedron with 5 faces. As the addition of one voxel is equivalent to the addition of two parallel planes in the dual space, after step q, the solution polyhedron has at more $2(2q+5)-4 = 4q+6$ vertices. As a matter of fact, the first loop of the function **Add_voxel** is done in $\mathcal{O}(q)$ time where q is the number of voxels of the piece of plane. Nevertheless, in practice, the number of vertices of B_q is much smaller than q.

In the loop, the first two tests can be done in constant time. The second loop does a new cover of the set B_q and is carried out in $\mathcal{O}(q)$ time. For the computation of the plane/line intersection, we saw that we need here to keep some particular knowledge on the values found for the intersection point, and we proposed in section 3.2 an algorithm that solves this problem in constant time. To recover the parameters of the solution planes, we will need after this algorithm a step to normalize the parameters (using Euclide's algorithm for instance to compute the gcd of the 3 denominators). This normalization can be done either for each B_q, or only at the end, for the vertices of $\bar{S}$.

For the function **Add_voxel**, we finally find a $\mathcal{O}(q^2)$ time complexity, where q is the number of voxels of the piece of plane.

5.2 Decompose-discrete-surface time complexity

Let us analyze line by line how this algorithm runs. Let n be the number of voxels which have a surfel belonging to the object surface. As a voxel has six faces, the first loop (line (1)) is done exactly 6 times. The second loop (line (2)) is run n times as we have n surface voxels. All the tests and instructions done between line (3) and line (8) run in constant time.

The time complexity of the loop line (9) depends on the maximum number of elements in `To-Process`.

Proposition 2 *At step number q (after the q first voxels) the maximum number of elements in* `To-Process` *is $4q+4$.*

Proof: After the processing of the first surfel, we put its 8 neighbors in `To-Process`. Moreover, we have seen in section 4 that any surfel belonging to a piece of plane has at least 3 neighbors in this plane. This means that at any time during the algorithm, each surfel of `To-Process` has at least two neighbors in this list. During the treatment of one surfel of the list, we delete this element from the list and we add its 8 neighbors. But, since at least 3 of them are already in the list, we add at most 5 for its neighbors. Finally, we add at most $5-1=4$ surfels at each step. Hence, at step number q, this list has at most $8+4(q-1)=4q+4$ elements. □

So, for the recognition of a naive plane with q voxels, this loop will be done at most $4q+4$ times. The choice in line (10) can be done in constant time, and in line (11), saving B needs a cover of the set B, which is done in $\mathcal{O}(q)$ time for a plane with q voxels. Moreover, for a naive plane with q voxels, the function **Add_voxel** runs in $\mathcal{O}(q^2)$ time, and the loop line (12) in $\mathcal{O}(8q^2) = \mathcal{O}(q^2)$ time. All the tests and instructions done between line (14) and (18) run in constant time. The restitution of B line (19) is done in $\mathcal{O}(q)$ time as it needs a cover of B_{save}. Then, we have all the elements to compute the global time complexity of this algorithm as a function of n, the number of voxels which have a surface surfel, and p, the size of the biggest recognized piece of plane. We get:

$$6n \times p \times (2p + 8p^2)$$

which leads to a final time complexity $\mathcal{O}(np^3)$.

6 Study on the voxel processing order

During the execution of the algorithm **Decompose-discrete-surface**, many choices have to be made concerning the order to process the voxels. They have an influence on the final decomposition we get: a given set of choices induces a different decomposition. Therefore, a study is useful to know if there exists a strategy leading to a "better" decomposition. In this section, we study this influence, comparing the results obtained with different strategies.

In the algorithm, three main choices are made for the tracking order. Indeed, in line (1), (2), (10) and (12), no details are given concerning the processing order for these different steps. But we can easily see that the choice made in line (10) does not influence the result: since our approach is surfel based, the recognition done for one direction has no influence on the recognitions done for the others. Then, three choices remain:

- the origin of each piece of plane (line (2));
- the next voxel to process during the recognition of a piece of plane (line (10));
- the tracking order of the 8 neighbors of a given voxel which determines the structure of the list `To-Process` (line (16)).

In this study, we give an insight in the influence of the last two ones.

First, we can notice that the order we process the 8 neighbors of a given voxel determines the order in which those neighbors are inserted into the list `To-Process`. Hence, the planes growing shape depends on two interdependant choices.

In the following, we present various strategies defined from those 2 choices.

6.1 Different strategies

The first strategy is also the simplest one to implement. In figure 6, we present first the 8 neighbors tracking and then the propagation scheme depending on which surfel we choose in the list of surfels `To-Process`. The numbers on the surfels refer to the order in which they are added in the list `To-Process`. With this first order, taking the last element of the list at each step leads to a very linear propagation scheme. This induces a main direction for the planes propagation. In fact, for any neighborhood tracking, choosing the last element of the list leads to a main direction given by the position of the last element processed during the 8 neighborhood tracking. If we take the first element of the list as a following surfel, we get the propagation drawn in figure 6. With this tracking, the left-down corner is always treated before the other sides, and the expansion is not regular nor isotropic.

Figure 7 illustrates a second strategy. The 8-neighbors tracking is now a clockwise tracking around the processed voxel (any other tracking around the voxel gives symmetrical results). The propagation obtained with the choice of the first surfel of the list is more isotropic than the previous one, even if the left-down corner is still processed first in an irregular way when we get further from the plane origin.

The main problem with those two strategies is that it is difficult to handle exactly the propagation even close to the origin.

A third method is illustrated in figure 8. This 8 neighbors tracking processes the voxels that are closer to the origin of the piece of plane first: the four 4-neighbors are first processed, and then the four 8-neighbors. As we saw that choosing the last element of the list induces linear propagations, we just show here the propagation obtained with the choice of the first element. We see that even after a big number of steps, the propagation scheme is always the same: the four directions ("sides") are processed one after the other in the clockwise direction. During the processing of one side, the surfels are processed according to their distance to the origin. After the processing of the 4 sides, the 4 corners are treated. So, the propagation is perfectly defined in this case, and is isotropic as each direction is processed in the same way as another, even if one direction is processed first.

6.2 Comparison results

In the following, we give some results for the comparison of the 3 tracking orders presented previously. To do so, we use the following criterion and objects: since a sphere is a symmetric object in all the directions, it would be nice to get pieces of planes that have nearly the same size. Hence, for a sphere, the standard deviation/average for the size of the recognized pieces of places should be as small as possible.

In the rest of this section, we denote *order* 1 (resp. 2, 3) the one which corresponds to the first (resp. second, third) strategy on the previous section, independently of the choice of the next voxel to process.

Figures 9 and 10 present the two comparisons we propose. The curves depicted are spline approximations of the discrete results.

In the first comparison (figure 9), each diagram represents the curves for one given tracking order, and each curve is the result choosing the first or the last voxel of the list. For all the strategies, the general shape of the curves is chaotic. This is due to the discrete nature of the data. Nevertheless, the curves have similar behaviors: for instance, all the curves have a local maximum when the radius is 5 or 8. It is quite easy to see that on those three first graphs, the curve corresponding to the choice of the last voxel of the list is globally worse than the one corresponding to the first voxel of the list. This suggests that the more isotropic the growing shape is, the better the result is.

The results of the second comparison are depicted in figure 10. In this case, the first voxel of the list is chosen and the comparison is done over the three different orders. This figure shows that the three curves cross over keeping very close values for any sphere radius. Hence, we cannot deduce from this graph that one tracking order is better than another one, and according to the very similar values we obtain, this suggests that those three orders have nearly the same behavior. Finally, it seems that the tracking order chosen does not have a very important influence on the result quality provided that it does not lead to a linear growing shape.

Nevertheless it would be interesting to see if the global behavior becomes stable when the radius of the sphere increases up to huge values, i.e. if one tracking order becomes better than the others, or if the curves always cross whatever the radius is. The treatment of very huge objects leads to implementation problems: indeed, since we work with integer fractions in the dual space, we quickly get some very long integers. The solution is to use a library to handle integers with infinite precision and this work is now in progress.

7 Results

In this section, we present some results about speed performances and images resulting of our algorithm.

7.1 Performance results

We did some tests for performance results on a Linux OS with a 1.8GHz processor. The algorithm is implemented in C++ with no particular optimizations. The figure 11(a) shows the results obtained for cubes of different sizes. In this figure, we uses logarithmic scales for the two axes. Hence, if the processing time depends directly on a power of the size of the object, the graph is a straight line. Moreover we only consider the time spent for the recognition of the pieces of planes, not including the input/output and display operations.

As the tracking order does not influence the result for a cube (the 6 faces are always found), we choose the tracking order that minimizes the lists tracking in the algorithm, i.e. the first order with the choice of the last element of the list `To-Process`. We can moreover notice that even if choosing the first element of the list `To-Process` induces one more list tracking in the time complexity computation, in practice, this choice has no effect on performance results.

Using section 5 results, we can evaluate the time complexity for a cube of side n: the number of surface surfels is of $\mathcal{O}(n^2)$ and the size of the biggest plane is of $\mathcal{O}(n^2)$ too, which leads to a $\mathcal{O}(n^8)$ theoritical bound for the time complexity. We see in figure 11 that the graph is really close to a straight line. In fact, if we consider the uncertainties due to such measurements, this result approaches very well a straight line with slope 3.5. This means that for the cube, the algorithm runs in $\mathcal{O}(n^{3.5})$ if n is the side of the square, which is quite better than the theoretical bound found in section 5.

We did the same job for a sphere, and the results are presented in figure 11(b). With this object, it is harder to do a comparison with the theoritical bound: indeed, it depends on the size of the biggest plane recognized, and it is hard to find a relation between the radius of the sphere and the size of the biggest plane. Nevertheless, the number of surface voxels is in $\mathcal{O}(n^2)$ if n is the radius of the sphere, and we can also suppose that the size of the biggest plane is a fraction of n^2. All together, we find a theoritical time complexity in $\mathcal{O}(n^8)$. Finally, it is interesting to notice that the curve we obtain is, as for the cube, very close to a straight line of slope 4.5. This means that the algorithm runs in $\mathcal{O}(n^{4.5})$ which is better than the estimation of the theoritical bound.

7.2 Image results

To finish, we give here some image results of this algorithm. For all the images presented here (Figures 12 and 13), each color corresponds to one piece of plane. For the visualization, if one surfel belongs to many pieces of planes, we display the color of the piece of plane that was recognized first.

Figure 12 presents some created and simple objects: one pyramid, a cube, a cube rotated in the grid and a chamfer cube (a cube of which one vertex has been cut by a plane). On the pyramid, we see that 4 planes have been recognized, for the 4 faces of the pyramid. All those planes are the same by symmetry: many voxels belong to two or more planes, and thus, with the priority rules we defined above, the plane first recognized is bigger than the others on the picture. Our algorithm recognizes the six faces of a cube for any rotation in the grid, and for a chamfer cube, it recognizes in addition the plane that cuts a vertex of this cube. As for the pyramid, the priority rules hide a big part of the sectionning plane: typically in this example, as the slopes of this plane and the face of the cube are close, the overlap between these planes is about 35 voxels.

Figure 13 gives the results for real objects: one image of a single hand's bones; one image of a piece of vertebra with high resolution. A table of the sizes of the planes recognized on the vertebra is presented in appendix A.

8 Conclusion and future work

In this paper, we have presented a new polynomial algorithm in the number of surface voxels to decompose the surface of any discrete volume into pieces of digital naive planes. To do so, we used an incremental naive plane recognition algorithm and we have shown some properties on the dual space associated to each piece of plane.

Using a 8-neighborhood voxels tracking, this decomposition algorithm forbids too long and narrow pieces of planes, and we analyzed some shape properties of the recognized pieces of planes. Then, we analyzed the global time complexity of this algorithm finding a polynomial bound depending on the number of surface voxels. A sharper analysis of this algorithm led us to study the influence of the different voxels tracking orders. In a last part, we made some performance tests on cubes of increasing side. These tests have shown that for the cube, practical performances are much better than the theoretical time complexity. The last images illustrated the position of the recognized pieces of plane for generated and real objects.

This work opens many future prospects, both on theoritical and practical aspects. First, some practical work can be done to improve performances: the use of a library that handles integers with arbitrary precision will enable to run this algorithm on bigger volumes.

On the theoretical side, it would be interesting to study more in details the structure of the dual space for a piece of plane as it has been done in 2D for discrete line segments [16].

Finally, this paper presented the first step of a more global goal that consists of finding a reversible polyhedrization of any discrete volume. To get such a polyhedrization, we need to transform each recognized piece of plane into a discrete polygon, a definition of which has been proposed in [17]. This supposes that we can define and place all the edges and the vertices between the found pieces of plane.

References

[1] W.E. Lorensen and H.E. Cline. Marching cubes : A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4):163–169,

1987.

[2] C.E. Kim and A. Rosenfeld. Convex digital solids. *IEEE Trans. on Pattern Anal. Machine Intell.*, PAMI-4(6):612–618, 1982.

[3] C.E. Kim and I. Stojmenović. On the recognition of digital planes in three dimensionnal space. *Pattern Recognition Letters*, 32:612–618, 1991.

[4] Ph. Borianne and J. Françon. Reversible polyhedrization of discrete volumes. In *Discrete Geometry for Computer Imagery*, pages 157–168, 1994.

[5] Isabelle Debled-Rennesson. *Etude et reconnaissance des droites et plans discrets.* PhD thesis, Université Louis Pasteur, Strasbourg, France, 1995.

[6] I. Debled-Rennesson and J.-P. Reveillès. An incremental algorithm for digital plane recognition. In *Discrete Geometry for Computer Imagery*, pages 207–222, 1994.

[7] Laurent Papier. *Polyédrisation et visualisation d'objets discrets tridimensionnels.* PhD thesis, Université Louis Pasteur, Strasbourg, France, 1999.

[8] L. Papier and J. Françon. Polyhedrization of the boundary of a voxel object. In Couprie Bertrand and Perroton, editors, *Discrete Geometry for Computer Imagery*, number 1568 in LNCS, pages 425–434. Springer, 1999.

[9] J. Burguet and R. Malgouyres. Strong thinning and polyhedrization of the surface of a voxel object. In G. Borgefors, I. Nyström, and G. Sanniti di Baja, editors, *Discrete Geometry for Computer Imagery*, number 1953 in LNCS, pages 222–234. Springer, 2000.

[10] J.-P. Reveillès. *Géométrie discrète, calcul en nombres entiers et algorithmique.* PhD thesis, Université Louis Pasteur, Strasbourg, France, 1991.

[11] E. Andres, R. Acharya, and C. Sibata. Discrete analytical hyperplanes. *Graphical Models and Image Processing*, 59(5):302–309, 1997.

[12] J. Vittone and J.-M. Chassery. Recognition of digital naive planes and polyhedization. In *Discrete Geometry for Computer Imagery*, number 1953 in LNCS, pages 296–307. Springer, 2000.

[13] Joëlle Vittone. *Caractérisation et reconnaissance de droites et de plans en géométrie discrète.* PhD thesis, Université Joseph Fourier, Grenoble, France, 1999.

[14] F. P. Preparata and M. I. Shamos. *Computational Geometry : An Introduction*. Springer-Verlag, 1985.

[15] D.J. Grabiner. Farey nets and multidimensionnal continued fractions. *Monath. Math.*, 114(1):35–61, 1992.

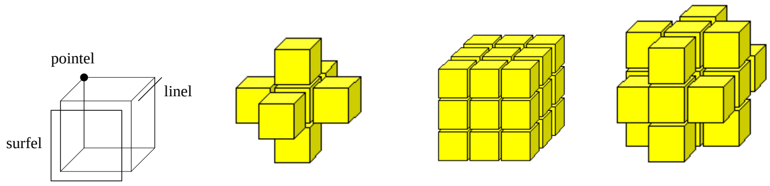
[16] M.D. McIlroy. A note on discrete representation of lines. *AT&T Technical Journal*, 64(2):481–490, February 1984.

[17] E. Andrès. Defining discrete objects for polygonalization : the standard model. In Lachaud Braquelaire and Vialard, editors, *Discrete Geometry for Computer Imagery*, number 2301 in LNCS, pages 313–325. Springer, 2002.

A Table of the plane sizes for the vertebra

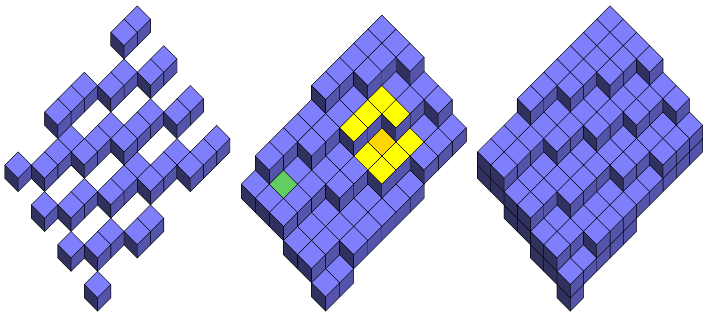
p. n.	size	p. n.	size	p. n.	size	p. n.	size
0	59	1	24	2	12	3	24
4	32	5	21	6	31	7	119
8	124	9	25	10	27	11	9
12	15	13	12	14	18	15	9
16	74	17	15	18	40	19	9
20	106	21	15	22	41	23	18
24	12	25	16	26	15	27	15
28	9	29	9	30	16	31	23
32	20	33	15	34	9	35	18
36	119	37	95	38	58	39	87

Figure 1: Table of the sizes of the planes for the vertebra (cf Figure 13): p.n. means plane number.



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

Figure 2: A voxel and the three classical neighborhoods



(a) (b) (c)

Figure 3: A discrete plane: $0 \leq 6x+13y+27z < \omega$ with different thicknesses: (a) $\omega = 15$ a thin plane with holes; (b) $\omega = 27$ a naive plane; (c) $\omega = 46$ a standard plane. A tricube is also depicted onto the naive plane.

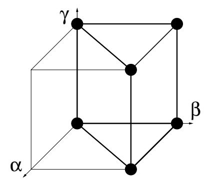


Figure 4: The initial set of solutions

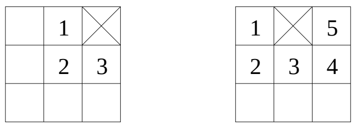


Figure 5: A surfel of a piece of plane has at least 3 neighbors in this plane

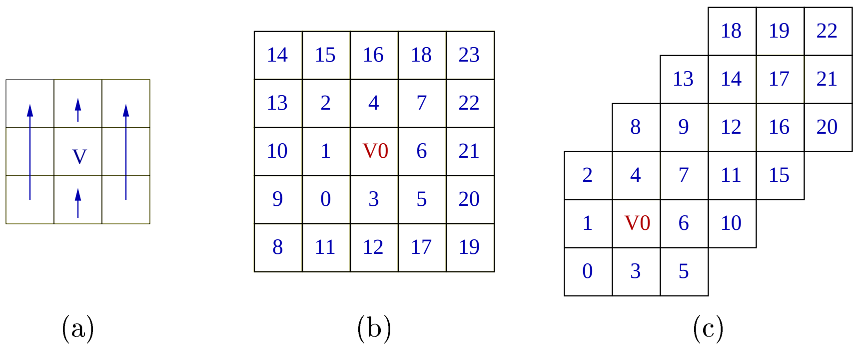


Figure 6: Strategy 1: (a) the 8 neighbors tracking; (b) propagation with the first element of the list `To-Process`; (c) propagation with the last element

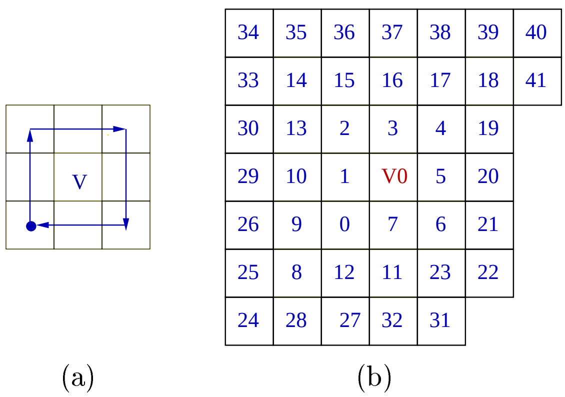
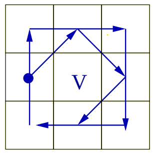
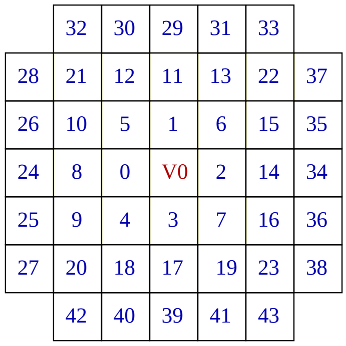


Figure 7: Strategy 2: (a) the 8 neighbors tracking; (b) propagation with the first element of the list `To-Process`; (c) propagation with the last element

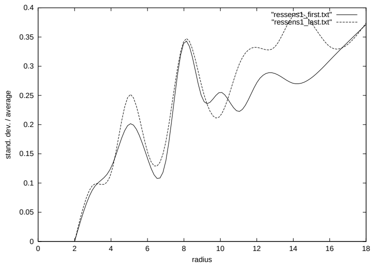


(a)

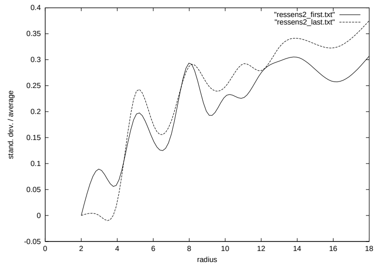


(b)

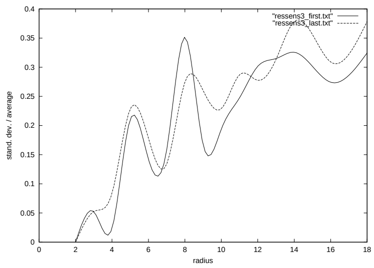
Figure 8: Strategy 3: (a) the 8 neighbors tracking; (b) propagation with the first element of the list `To-Process`



(a)



(b)



(c)

Figure 9: Comparison for the choice of the next voxel to process: (a) order 1; (b) order 2; (c) order 3

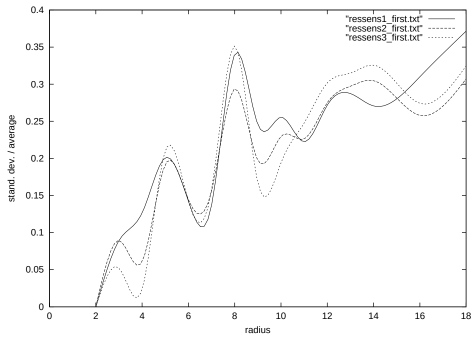
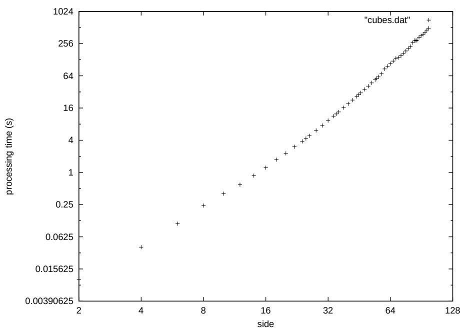
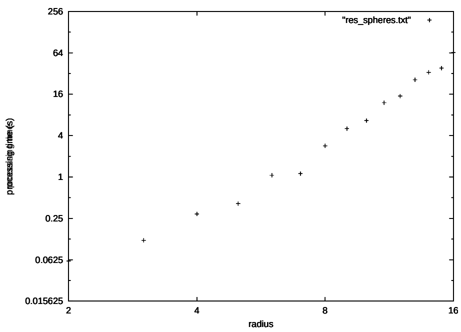


Figure 10: Comparison for the 8-neighbors tracking order

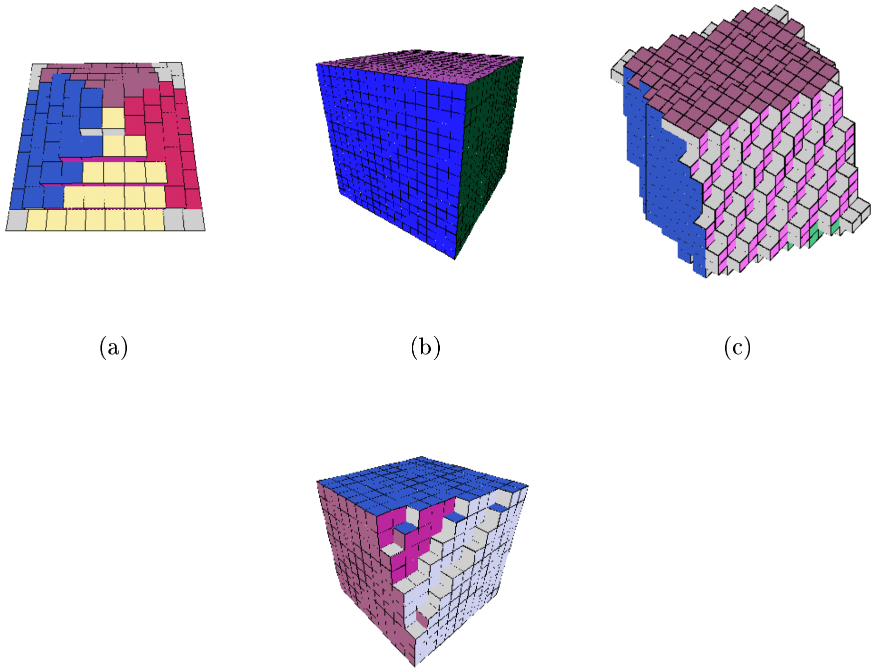


(a)



(b)

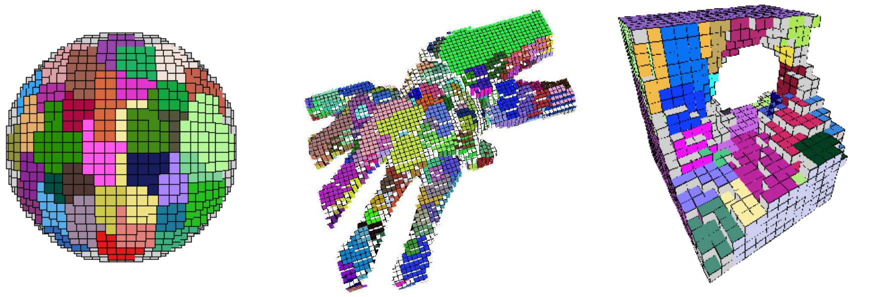
Figure 11: Performance results: (a) for the cube; (b) for the sphere



(a) (b) (c)

(d)

Figure 12: Simple objects: (a) A pyramid with basis 10 and height 5; (b) A cube of side 16; (c) A cube rotated in the grid; (d) A chamfer cube



(a) (b) (c)

Figure 13: (a) A sphere of radius 14; (b) A hand image; (c) A small part of a human vertebra