Repairing 3D binary images using the BCC grid with a 4-valued combinatorial coordinate system

POST-PRINT

Article published on *Information Sciences*, Vol. 499, 2019, pp.47-61
https://www.sciencedirect.com/science/article/pii/S0020025518301348?via%3Dihub

Lidija Čomić
Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia
comic@uns.ac.rs

Paola Magillo
Department of Computer Science, Bioengineering, Robotics, and Systems Engineering,
University of Genova, Genova, Italy magillo@dibris.unige.it

Abstract

A 3D binary image I is called well-composed if the set of points in the topological boundary of the cubes in I is a 2-manifold. Repairing a 3D binary image is a process which produces a well composed image (or a polyhedral complex) from the non-well-composed image I.

We propose here to repair 3D images by associating the body-centered cubic grid (BCC grid) to the cubical grid. The obtained polyhedral complex is well composed, since two voxels in the BCC grid either share an entire face or are disjoint. We show that the obtained complex is homotopy equivalent to the cubical complex naturally associated with the image I.

To efficiently encode and manipulate the BCC grid, we present an integer 4-valued combinatorial coordinate system that addresses cells of all dimensions (voxels, faces, edges and vertices), and allows capturing all the topological incidence and adjacency relations between cells by using only integer operations.

We illustrate an application of this coordinate system on two tasks related with the repaired image: boundary reconstruction and computation of the Euler characteristic.

1 Introduction

It is well known that in the 3D cubical grid, different adjacency relations between the cubes can be used: face, edge and corner adjacency (6-, 18- and 26-adjacency). The topological properties of the image may be dramatically affected by the choice of the adjacency relation. To maintain some similarity between discrete and continuous topology, usually a pair of different adjacency relations is used for object (black) and background (white) cubes.

An image I is called *well-composed* if the set of points in the topological boundary of the cubes in I is a 2-manifold. Well-composed images have advantageous topological properties compared to the non-well-composed ones: boundary reconstruction and thinning algorithms are simpler [Kon1989] and ray casting algorithms are guaranteed to create no false holes in the voxelized surface [Coh1997], to name just a few. The process of transforming a given image in a well-composed one, that is in some sense similar to the original, is called *repairing*.

Here, we address image repairing by using the Body-Centered Cubic (BCC) grid. The contributions of this paper are:

- A simple repairing procedure, which transports the given image from the cubical to the BCC grid. The result is a well-composed image which is homotopy equivalent to the original one. Furthermore, we show how it can be displayed in a visually similar way.

- A 4-valued integer coordinate system for the BCC grid, which supports our repairing procedure. This coordinate system is compact, and allows addressing all cells of a BCC grid, and retrieving all the topological adjacency and incidence relations between the cells, by using just integer arithmetics.
- Algorithms for boundary reconstruction and computation of the Euler characteristic of the repaired image, implemented over the BCC grid through our coordinate system. This also demonstrates that our 4-valued coordinate system supports the design of algorithms in an elegant and efficient way.

2 Related Work

Well-composedness of a 3D image I can be checked by examining local configurations of 2×2 and $2 \times 2 \times 2$ cubes in I [Lat1997]. Forbidden configurations correspond to an edge or a vertex having exactly two incident black (or white) cubes, where such two cubes do not share a face. A closely related notion is that of a gap. Different formulae for the number of 0-gaps in 2D [Bri2006, Bri2005, Mai2011], 1-gaps in 3D [Mai2013] and $(n-2)$-gaps in nD binary images [Bri2009] have been proposed.

There have been several approaches in the literature to repair a 3D binary image I and the associated cubical complex Q. We review in more detail only those [Gon2015, Ros1998, Siq2008] that work on the initial cubical complex, without passing to the dual complex (like e.g. Lachaud and Montanvert [Lac2000]). The well-composed complex produced by the reviewed algorithms is either cubical [Ros1998, Siq2008] or polyhedral [Gon2015], that may [Gon2015, Ros1998] or may not [Siq2008] be homotopy equivalent to Q, and may [Gon2015, Ros1998] or may not [Siq2008] have increased memory requirements. The algorithm that we propose here produces a polyhedral complex that is homotopy equivalent to Q, uses just one type of voxels, and the implementation allows encoding all the cells and their topological relations in only twice as much space as Q.

One of the first repairing algorithms was proposed by Rosenfeld et al. [Ros1998] for 2D binary images. It uses a rescaling by factor 3 in both x and y axes of a digital grid. In the rescaled grid, changing all black pixels involved in a critical configuration to white pixels (or vice versa) removes all critical configurations, without creating other such configurations. The repaired image is homotopy equivalent to the initial image with the appropriate adjacency relation. The method could be extended to 3D, but would require a 27 times larger image [Siq2008].

The randomized algorithm proposed by Siqueira et al. [Siq2008] iteratively changes white cubes to black ones, until a well-composed image is obtained. The grid resolution of the repaired image is the same as that of the input image, but there is no guarantee that the topology (homotopy) of the input image is preserved. The randomized algorithm was extended to make well-composed gray-scale images (functions) [Bou2015] (such that all the lower and upper level sets of the function are well-composed).

A recent algorithm proposed by Gonzalez-Diaz et al. [Gon2015] creates a polyhedral well-composed complex homotopy equivalent to the input cubical complex. The idea is to thicken the neighborhood of critical vertices and edges and subdivide it into polyhedra. The exact transformation of the input cubical complex is detailed for each of the 11 types of critical configurations in $2 \times 2 \times 2$ blocks of cubes. The input image is rescaled by factor 4 in all three dimensions in order to store the obtained polyhedral complex through integer coordinates, and all the topological relations between its cells that are necessary for (co)homology computation. There are 27 different types of polyhedra used in the repairing process. Later [Gon2017], a minimal encoding of the polyhedral complex was proposed, that would allow dividing the polyhedron coordinates by 2. A procedure to extract the boundary of the polyhedral complex was also developed.

Our repairing method is somehow similar to this one, in the sense that we also thicken the complex near critical vertices and edges, but we do this by passing from the cubical to the BCC grid. A more detailed comparison of the two approaches will be provided in Section 7.5. Our repairing algorithm is simple and elegant, and produces a complex Γ in the BCC grid, which is well-composed by construction, thanks to the properties of the BCC grid. We show that the repaired complex Γ is homotopy equivalent to the initial cubical complex Q of I. Thus, it can be used for homology computation, and specifically for the computation of the Euler characteristic.

These and other applications require an easy access to the cells of all dimensions in the complex and a straightforward retrieval of their topological relations. In our past work, we have developed a 3-valued compact coordinate system for the BCC grid [Com2016], with three independent Cartesian coordinates, that addresses all cells in the grid. Here, we present an equivalent symmetric 4-valued coordinate system, with four dependent coordinates. It can be viewed as a translation of the combinatorial 3-valued coordinate system [Com2016] to a space with four dependent generators instead of three independent Cartesian ones. It can equivalently be viewed as an extension of the coordinate system, that defines the coordinate values of the 3-cells (voxels) of the BCC grid [Nag2008], to address all cells, and not only the voxels of the grid. We believe that this interpretation is more suited to the present task, where vertices of the cubical complex, located in the direction of the four dependent axes, play an important role in the repairing process. As usual in this type of coordinate systems, all incidence and adjacency relations between cells can be expressed in terms of cell coordinates only, through simple integer operations, without the use of additional data structures or pointers.

3 Background Notions

We give some basic notions on well-composed 3D images [Lat1997] and the BCC grid [Kit2004].

3.1 Well-Composed Images

A *binary image* I [Kle2004, Kon1989] is a finite collection of (black or object) cubes in the cubical grid. The cubes in the complement I^c of I are called white (background). The associated cubical complex Q is a cell complex that consists of the cubes in I together with all of their square faces, edges and vertices. Its carrier $|Q|$ is the set of points in $\mathbb{R}^3$ that belong to some cell in Q.

An image $I \subset \mathbb{Z}^3$ is *well-composed* if and only if its boundary is a 2-manifold [Lat1997]. Two types of *critical* local configurations can be defined:

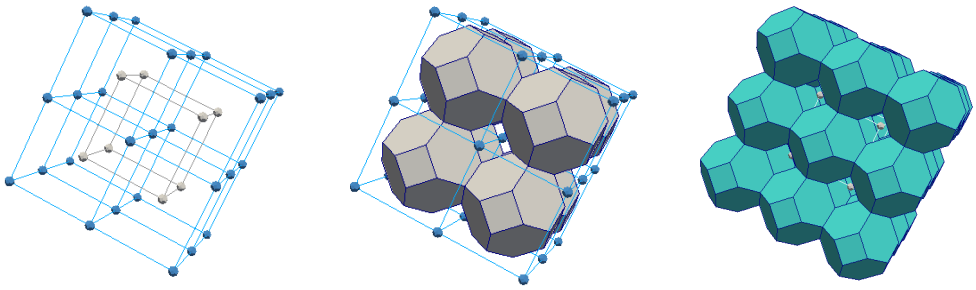
- 2×2 configurations with two diagonal edge-adjacent cubes in I and the other two in I^c (*critical edge*), and
- $2 \times 2 \times 2$ configurations with two diagonal vertex-adjacent cubes in I and the other six cubes in I^c, or vice versa (*critical vertex*).

An image $I \subset \mathbb{Z}^3$ is well-composed if and only if there is no occurrence neither of the first type nor of the second type of critical configurations in I. Among the different configurations of the eight cubes around a vertex, there are eleven configurations featuring critical vertices and edges [Gon2015, Mor1980].

3.2 The BCC Grid

The body-centered cubic grid (BCC grid) is a Voronoi tessellation of $\mathbb{R}^3$ associated with points in $\mathbb{Z}^3 \cup (\mathbb{Z}+1/2)^3$. The centers of Voronoi regions are points with three integer Cartesian coordinates or with three half-integer Cartesian coordinates. The BCC grid describes the placement of atoms in various metals (e.g. sodium, chromium, vanadium, tungsten) and ionic crystals (e.g. cesium chloride) [Kit2004]. One atom is placed at each corner of a unit cubic cell and, additionally, one atom is placed at the center of each unit cube (see Figure 1 (a)). The voxel (3-cell) of the BCC grid is a truncated octahedron or, equivalently, a truncated hexahedron (cube), with eight regular hexagonal faces and six square faces, 36 edges and 24 vertices (see Figure 1(b,c)).

The BCC grid, like the cubic one, has the structure of an abstract cell complex [Kov2008], with naturally defined boundary and dimension. It is composed of two interlaced cubic grids, where voxels of the first (second) grid are centered at points with integer (half-integer) Cartesian coordinates (see Figure 1). We say a voxel is *even* (*odd*) if the coordinates of its center have integer (half-integer) values. Remember that the BCC grid is a lattice, and all voxels are identical. We use the differentiation of even and odd voxels only for identifying the cubic grid they come from.



(a) (b) (c)

Figure 1: (a) Two interlaced cubic grids, one grid with centers in points with integer coordinates (blue) and the other grid with centers in points with half-integer coordinates (gray). (b) Voxels of the odd grid. (c) Voxels of the even grid.

There are two types of neighborhood relation between two voxels of the BCC grid: either they share a hexagonal face, and are called *hex-adjacent* (or 1-neighbors), or they share a square face, and are called *quad-adjacent* (2-neighbors). Two hex-adjacent voxels are one even and one odd, while two quad-adjacent voxels are of the same type: they are both even or both odd. Each voxel is hex-adjacent to eight, and quad-adjacent to six voxels. Note that the set of four voxels of the same type in a 2×2 block is not simply connected, as opposed to such set of four cubes (see Figure 1 (b)).

There are no critical configurations in the BCC grid, as two voxels either share an entire face or are disjoint. Thus, any set of voxels in the BCC grid is well-composed.

4 Repairing 3D Binary Images

We describe in detail our algorithm for repairing a cubical complex Q, and we show that the produced complex Γ is homotopy equivalent to Q.

4.1 Algorithm

The cubical grid is a tessellation of the space $\mathbb{R}^3$ into Voronoi regions associated with the points in $\mathbb{Z}^3$ (points with three integer Cartesian coordinates). We will consider also vertices of the cubes in the cubical grid (points with half-integer Cartesian coordinates). By tessellating the space $\mathbb{R}^3$ into Voronoi regions associated with points in $\mathbb{Z}^3 \cup (\mathbb{Z}+1/2)^3$ we obtain the BCC grid, in which an even voxel of the BCC grid is associated with each cube, and an odd voxel is associated with each vertex of the cubical grid.

We propose to repair a cubical complex Q by associating with it a polyhedral complex Γ in the BCC grid. The motivation for passing to the BCC grid is twofold: (1) each set of voxels in the BCC grid is well-composed and (2) the classification of critical configurations in Q is done with respect to the vertices in the cubical grid, which are naturally associated with the odd voxels in the BCC grid.

For each cube in Q, we include the corresponding even voxel in Γ. We include also the odd voxels that are necessary for topology preservation. The included odd voxels are associated with:

(i) Central vertices of critical configurations (critical vertices and vertices incident to critical edges), that are not incident to a white edge (an edge surrounded by four white cubes) (Figure 2 (i)); they correct the non-manifoldness of the vertex.

(ii) Vertices incident to one critical edge and one white edge, if the white edge is in the direction of positive x, y or z axis from the vertex (Figure 2 (ii)); they correct the non-manifoldness of the critical edge.

(*iii*) Vertices incident to a black edge (an edge surrounded by four black cubes), if the black edge is in the direction of negative x, y or z axis from the vertex; they prevent the creation of holes formed by four black cubes around the black edge (Figure 2 (iii)).

Note that by rule (*iii*), the odd voxels corresponding to interior vertices of Q (vertices surrounded by eight black cubes) are also included in Γ. This rule ensures that no hole is generated in a 2×2 block of black cubes (see Figure 1 (b)).

Thus, we add odd voxels associated with all interior vertices, all critical vertices that are not incident to a white edge, at least one vertex incident to a black edge (to prevent the creation of holes around the black edge) and at most one vertex incident to a white edge (to prevent joining of different parts along the white edge). In the last two cases, an orientation is considered for edges in such a way that vertices are added only at the positive sides of the object with respect to the coordinate axes.

We illustrate these rules through some examples. Rule (*i*) includes odd voxels corresponding to critical vertices, and at least one of the two odd voxels corresponding to the endpoints of each critical edge. These voxels maintain the topology of Q with 26-adjacency. For example, the odd voxel corresponding to the central vertex in Figure 2 (*i*) connects the three even voxels corresponding to the three black cubes. The other odd voxel is included by rule (*ii*).

Our rules include enough odd voxels to prevent the creation of holes and cavities in Γ. Consider the two configurations in Figures 2 (ii) and (iii), where the red voxels in (c) are added by rule (*ii*) and (*iii*), respectively. If one of the two parts composing each image were translated to get glued with the other part, then the same two central odd voxels would be added (the vertex with two incident critical edges would now be added by rule (*i*). If they were not, then a through hole would be created in Figure 2 (ii), and a cavity in Figure 2 (iii).

Also, our rules ensure not to add too many odd voxels, that would create new connections which did not exist in the cubic image. This problem would arise, e.g., if we added both endpoints of the critical edge in Figure 2 (ii), or of the black edge in Figure 2 (iii).

4.2 Homotopy Equivalence

We show that the polyhedral complex Γ produced by the repairing algorithm is homotopy equivalent to the cubical complex Q. Recall that two polyhedral complexes Γ and Q are homotopy equivalent if the carrier $|\Gamma|$ of Γ can be continuously deformed into the carrier $|Q|$ of Q.

Proposition 1. *The repaired polyhedral complex Γ is homotopy equivalent to the cubical complex Q.*

Proof. We will construct the required continuous deformation by mapping the vertices of the complex Γ to $|Q|$, and extending the mapping linearly to the cells in Γ.

For each odd interior voxel V in Γ, centered at the vertex v of the cubical complex Q, we map all the vertices of voxel V to v.

For each odd boundary voxel V (an odd voxel that is adjacent both to a black and a white even voxel) centered at the vertex v, we make the following subdivision (the added vertices are illustrated in Figure 3 (a)):

- We insert a vertex d at the midpoint of each edge of V that is shared by a hex-face and a quad-face of V. We denote as c the midpoints of the remaining edges of V.
- We insert a vertex b at the barycenter of each hex-face of V.
- We insert the edges from b to each vertex (old or new) of the hex-face.
- We insert edges, faces and 3-cells from the center v of V to all vertices, edges and faces in the boundary of V, respectively.

We denote as Γ' the subdivision of Γ. The two complexes have the same carrier ($|\Gamma'| = |\Gamma|$). Let us denote as d' the barycenters of the six quad-faces of V (they are the midpoints of the edges in the cubical grid incident to v).

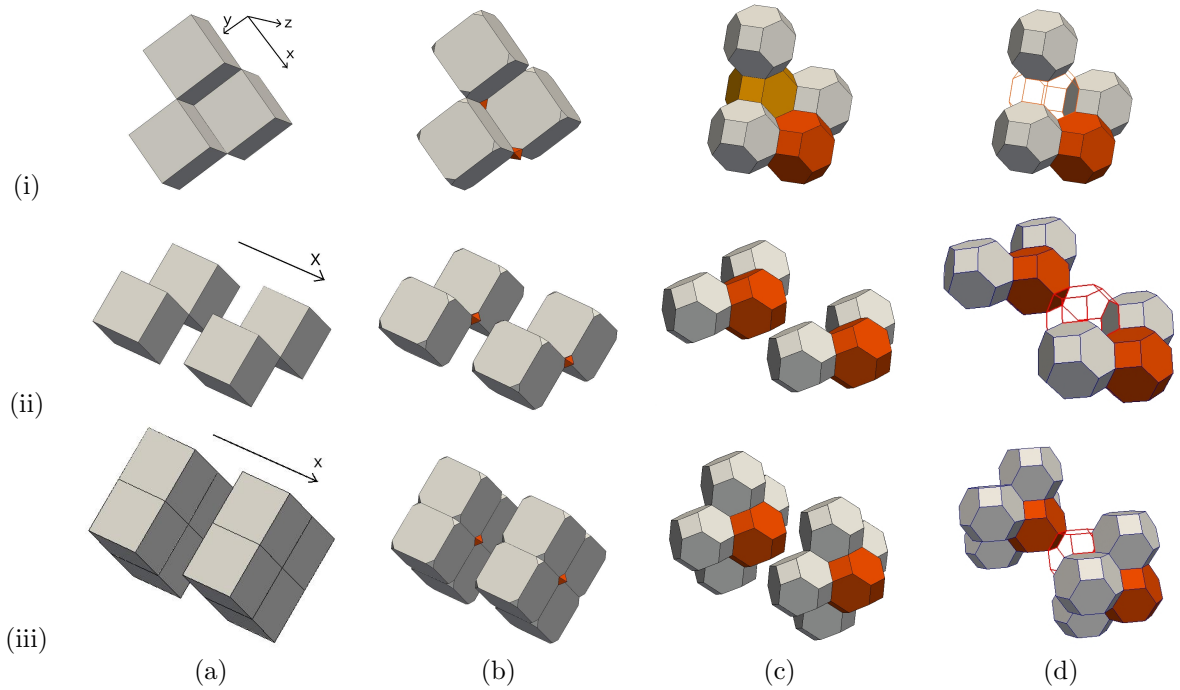


Figure 2: The three rules in the repairing algorithm. The cubical complex Q (a); vertices whose odd voxels are added are shown as red diamonds, and vertices whose odd voxels remain white are shown by truncating the cube at the vertex (b); the repaired complex Γ (c); the reason for imposing the rule (d).

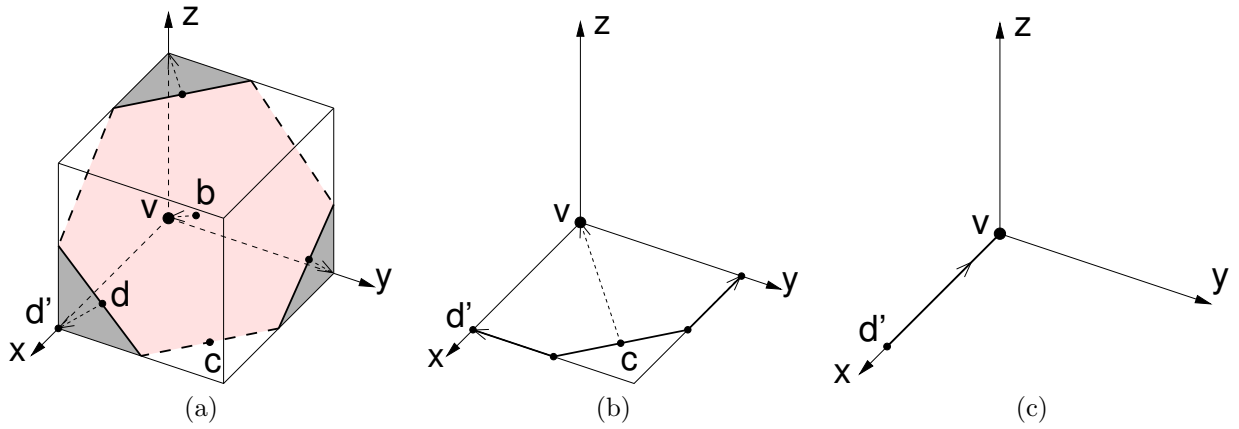


Figure 3: (a) Point b is mapped to v, points d are mapped to points d'. (b) Point c is mapped to v, endpoints of the edge containing c are mapped to points d'. (c) Point d' is mapped to v.

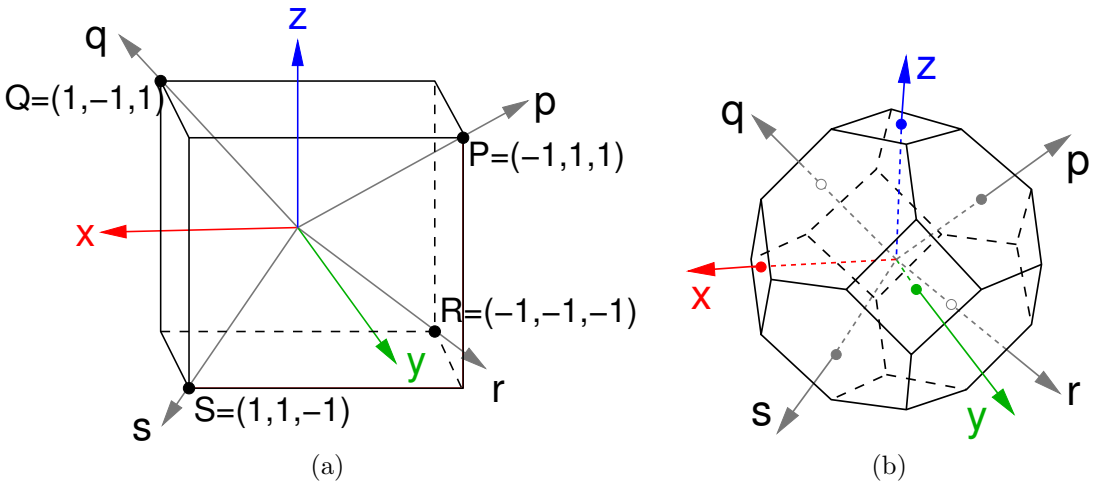


Figure 4: (a) Orientation of the four octahedral axes p, q, r, s; (b) Each face of a voxel is oriented according to the (positive or negative) axis direction: Cartesian axes for quad-faces and octahedral axes for hex-faces.

If the odd voxel V is in Γ (if V is black), we push the protruding parts of Γ' onto Q as follows:

- For each exposed hex-face of V (a face incident to a white even voxel B), map its barycenter b to v, map each point d of the hex-face to the closest of the points d' (see Figure 3 (a)).
- For each pair of adjacent even white voxels hex-adjacent to V, map the midpoint c of the edge shared by the two voxels and by the voxel V to v. Map each of the two endpoints of the edge to the closest point d' (see Figure 3 (b)).
- For each white edge e in the cubical grid incident to v, map the point d' on e to v (Figure 3 (c)). Our rule for including odd voxels in Γ guarantees that there will be no attempts to move d' to the other endpoint w of e.

If the odd voxel V is not in Γ (if V is white), we fill in the parts missing from Q. The vertex maps are defined in the same way as above, with white replaced by black.

□

5 Combinatorial 4-Coordinate System - Definition

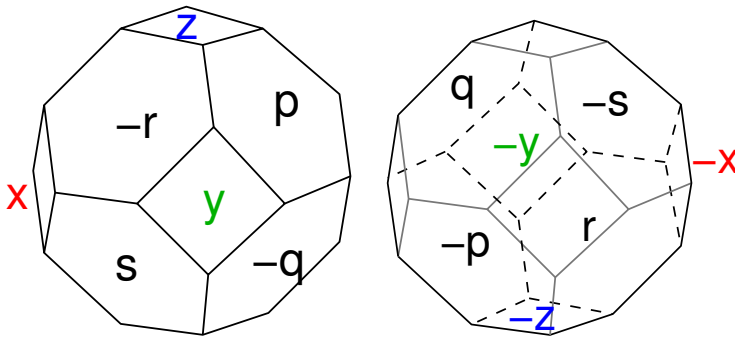
The combinatorial coordinate system proposed here elaborates from [Nag2008], in which each voxel (3-cell or octahedron) in the BCC grid is identified by four coordinates (p, q, r, s), where p, q, r, s are all even for even voxels, they are all odd for odd voxels, and $p+q+r+s = 0$ (see Figure 1). Figure 5 shows the orientation of the four axes p, q, r, s with respect to Cartesian axes x, y, z. We extend this coordinate system for voxels to address all cells in the BCC grid, including the lower-dimensional ones. In our coordinate system, all cells, of any dimension, have integer coordinate values that sum up to 0. In order to do this, we rescale the grid, so that a unit cell in each of the two interlaced cubic grids defining the BCC grid has size $4 \times 4 \times 4$.

The coordinate system proposed here can be viewed also as a transformation of the 3-valued combinatorial coordinate system for the BCC grid proposed in [Com2016] to the 4-valued one. The transformation is $p = 2(-x+y+z)$, $q = 2(x-y+z)$, $r = 2(-x-y-z)$, $s = 2(x+y-z)$, where x, y, z are three independent Cartesian coordinates and p, q, r, s are four dependent coordinates with axes pointing in the direction of the vertices of the unit cube centered at the origin. The vertices of cubical voxels play a prominent role in the repairing process, and the 4-valued version of the coordinate system is more suited for this purpose.

5.1 Voxels

A voxel has even coordinate values that sum up to 0. An even voxel has coordinates (p, q, r, s), such that $p, q, r, s \equiv 0$ or $\equiv 4 \pmod 8$, and $p + q + r + s = 0$. An odd voxel has coordinates (p, q, r, s), such that $p, q, r, s \equiv 2$ or $\equiv 6 \pmod 8$, and $p + q + r + s = 0$.

Each even (odd) voxel is hex-adjacent to eight odd (even) voxels that are obtained by adding ± 6 to one, and ∓ 2 to the other three of its coordinates. Each (even or odd) voxel is quad-adjacent to six (even or odd) voxels that are obtained by adding $+4$ to two, and -4 to the other two of its coordinates. Figure 5 illustrates the coordinates of voxels adjacent to even voxel $(0, 0, 0, 0)$.



(a)

axis	face	adj. voxel
$+x$	(-2,2,-2,2)	(-4,4,-4,4)
$+y$	(2,-2,-2,2)	(4,-4,-4,4)
$+z$	(2,2,-2,-2)	(4,4,-4,-4)
$-x$	(2,-2,2,-2)	(4,-4,4,-4)
$-y$	(-2,2,2,-2)	(-4,4,4,-4)
$-z$	(-2,-2,2,2)	(-4,-4,4,4)
$+p$	(3,-1,-1,-1)	(6,-2,-2,-2)
$+q$	(-1,3,-1,-1)	(-2,6,-2,-2)
$+r$	(-1,-1,3,-1)	(-2,-2,6,-2)
$+s$	(-1,-1,-1,3)	(-2,-2,-2,6)
$-p$	(-3,1,1,1)	(-6,2,2,2)
$-q$	(1,-3,1,1)	(2,-6,2,2)
$-r$	(1,1,-3,1)	(2,2,-6,2)
$-s$	(1,1,1,-3)	(2,2,2,-6)

(b)

Figure 5: (a) Visible and invisible faces of a voxel, with orientation. (b) Combinatorial coordinates for faces and adjacent voxels of even voxel $(0, 0, 0, 0)$.

5.2 Faces

A hex-face is shared by an even voxel $v_1 = (p_1, q_1, r_1, s_1)$, and a hex-adjacent odd voxel $v_2 = (p_2, q_2, r_2, s_2)$. The multiset of coordinate differences between the two voxels is $\{2, 2, 2, -6\}$ or $\{-2, -2, -2, 6\}$ (cfr. hex-adjacent voxels in Section 5.1). We assign as such hex-face coordinates the average $((p_1 + p_2)/2, (q_1 + q_2)/2, (r_1 + r_2)/2, (s_1 + s_2)/2)$ of the coordinates of the incident voxels.

If we consider separately the value of each coordinate p, q, r, s modulo 8 for each of the two voxels sharing the face, and we take all possible combinations into account, we can conclude that three coordinates of a hex-face are $\equiv 1$ and one is $\equiv 5 \pmod 8$ or vice versa, or three are $\equiv 3$ and one is $\equiv 7 \pmod 8$ or vice versa. The sum of hex-face coordinates is 0. For example, the common hex-face for even voxel $(0, 0, 0, 0)$ and hex-adjacent odd voxel $(2, 2, 2, -6)$ is $(1, 1, 1, -3)$. Figure 5 illustrates the coordinates of hex-faces incident to even voxel $(0, 0, 0, 0)$.

A quad-face is shared by two quad-adjacent even or two quad-adjacent odd voxels. We call such faces *even* and *odd quad-faces*, respectively. Let (p_1, q_1, r_1, s_1) and (p_2, q_2, r_2, s_2) be two quad-adjacent voxels. We assign to their common quad-face average coordinates $((p_1 + p_2)/2, (q_1 + q_2)/2, (r_1 + r_2)/2, (s_1 + s_2)/2)$. Of the two quad-adjacent even voxels, all coordinates of one voxel are $\equiv 0$, all coordinates of the other voxel are $\equiv 4 \pmod 8$, and the multiset of coordinate differences between the two voxels is $\{4, 4, -4, -4\}$. The sum of coordinates of both voxels is 0. Similarly as above, we conclude that two coordinates of an even quad-face are $\equiv 2$, and other two are $\equiv 6 \pmod 8$. The sum of even quad-face coordinates is 0. For example, even quad-face shared by quad-adjacent even voxels $(0, 0, 0, 0)$ and $(4, 4, -4, -4)$ is $(2, 2, -2, -2)$.

Of the two quad-adjacent odd voxels, all coordinates of one voxel are $\equiv 2$, all coordinates of the other voxel are $\equiv 6 \pmod 8$, and the multiset of coordinate differences between the two voxels is again $\{4, 4, -4, -4\}$.

The sum of coordinates of both voxels is 0. Thus, two coordinates of an odd quad-face are $\equiv 0$, and other two are $\equiv 4 \pmod 8$. The sum of odd quad-face coordinates is 0. For example, odd quad-face shared by quad-adjacent odd voxels $(2, 2, 2, -6)$ and $(6, -2, -2, -2)$ is $(4, 0, 0, -4)$.

Figure 5 illustrates the coordinates of even quad-faces incident to even voxel $(0, 0, 0, 0)$.

5.3 Edges

Each edge is incident to one quad-face and two hex-faces. The multiset of coordinate differences of the two incident hex-faces is $\{2, 2, -2, -2\}$. We take the arithmetic mean of the coordinates of the incident hex-faces and assign it to the edge. For example, two hex-faces $(1, 1, -3, 1)$ and $(3, 3, -5, -1)$ (and quad-face $(2, 2, -2, -2)$) determine the edge $(2, 2, -4, 0)$.

An edge has two coordinates $\equiv 0$ and two coordinates $\equiv 2 \pmod 4$. If the incident quad-face is an even quad-face, than both coordinates of the edge that are $\equiv 2 \pmod 4$ are either $\equiv 2$ or $\equiv 6 \pmod 8$. We call such edges *even edges*. If the incident quad-face is an odd quad-face, than one of the coordinates of the edge that are $\equiv 2 \pmod 4$ is $\equiv 2 \pmod 8$ and the other is $\equiv 6 \pmod 8$. We call such edges *odd edges*. These claims can be proven in the similar way as above for faces. We skip the details for brevity.

Alternatively, edge coordinates can be obtained from the coordinates of the incident voxels. An even edge is shared by an odd voxel $v_1 = (p_1, q_1, r_1, s_1)$ and two even voxels $v_2 = (p_2, q_2, r_2, s_2)$ and $v_3 = (p_3, q_3, r_3, s_3)$. The coordinates of the edge are equal to $(2p_1+p_2+p_3)/4, (2q_1+q_2+q_3)/4, (2r_1+r_2+r_3)/4, (2s_1+s_2+s_3)/4)$. For example, the even edge determined by quad-adjacent even voxels $(0, 0, 0, 0)$ and $(4, 4, -4, -4)$, and odd voxel $(2, 2, -6, 2)$ hex-adjacent to both is $(2, 2, -4, 0)$. Coordinates for an odd edge can be obtained in a similar manner.

Finally, edge coordinates can be obtained from the coordinates of the incident quad-face. An even quad-face has two coordinates $\equiv 2$ and two $\equiv 6 \pmod 8$. The four incident even edges are obtained by keeping two face coordinates that are $\equiv 2 \pmod 8$ and by increasing one of the remaining two coordinates and decreasing the other by 2, or by keeping the two face coordinates that are $\equiv 6 \pmod 8$ and by increasing one of the remaining two coordinates and decreasing the other by 2. For example, the four even edges incident to even quad-face $(2, 2, -2, -2)$ are $(2, 2, -4, 0)$, $(2, 2, 0, -4)$, $(4, 0, -2, -2)$ and $(0, 4, -2, -2)$.

The coordinates of odd edges incident to odd quad-face are obtained in a similar manner.

5.4 Vertices

Each vertex is shared by two even and two odd voxels. We assign to the vertex the average of the coordinates of the four incident voxels. All vertex coordinates are odd: one is $\equiv 1$, one is $\equiv 3$, one is $\equiv 5$ and one is $\equiv 7 \pmod 8$. Their sum is equal to 0.

For example, the vertex shared by two even voxels $(0, 0, 0, 0)$ and $(4, 4, -4, -4)$ and two odd voxels $(-2, 6, -2, -2)$ and $(2, 2, -6, 2)$ is $(1, 3, -3, -1)$.

Alternatively, vertex coordinates can be obtained as the average of coordinates of the four incident hex-faces and two incident quad-faces. For example, vertex $(1, 3, -3, -1)$ is incident to four hex-faces $(1, 1, -3, 1)$, $(-1, 3, -1, -1)$, $(3, 3, -5, -1)$ and $(1, 5, -3, -3)$, and to two quad-faces $(2, 2, -2, -2)$ and $(0, 4, -4, 0)$ (the first one is even and the second one is odd).

Vertex coordinates can also be obtained from the coordinates of the four incident edges, again by taking their arithmetic mean. For example, vertex $(1, 3, -3, -1)$ is incident to even edges $(2, 2, -4, 0)$ and $(0, 4, -2, -2)$ and odd edges $(2, 4, -4, -2)$ and $(0, 2, -2, 0)$.

6 Combinatorial 4-Coordinate System - Summary

We give a compact but detailed summary of the coordinate system and of the topological relations between the cells in the BCC grid expressed through the combinatorial coordinates.

In Table 1, we describe the coordinate values for different types of cells. For example, we see that an odd quad-face has two coordinates $\equiv 0$ and two $\equiv 4 \pmod 8$. The sum of coordinates for each cell is 0.

Each tuple of the form given in Table 1 represents a cell of the corresponding dimension, and each cell has a unique and unambiguous representation as a tuple.

Table 1: The numbers and possible values of the assigned coordinate values.

cell	number	coordinates (mod 8)
even voxel	4	$\equiv 0$ or $\equiv 4$
odd voxel	4	$\equiv 2$ or $\equiv 6$ (mod 4)
hex-face	3	$\equiv 1$ or $\equiv 3$ or $\equiv 5$ or $\equiv 7$
	1	$\equiv 5$ or $\equiv 7$ or $\equiv 1$ or $\equiv 3$ respectively
even quad-face	2	$\equiv 2$
	2	$\equiv 6$
odd quad-face	2	$\equiv 0$
	2	$\equiv 4$
even edge	2	$\equiv 2$ or $\equiv 6$
	1	$\equiv 0$
	1	$\equiv 4$
odd edge	2	$\equiv 0$ or $\equiv 4$
	1	$\equiv 2$
	1	$\equiv 6$
vertex	1	$\equiv 1$
	1	$\equiv 3$
	1	$\equiv 5$
	1	$\equiv 7$

In Tables 2, 3, 4, and 5, we describe the retrieval of topological relations through the coordinate system for voxels, hex- and quad-faces, even edges and vertices). In the tables, we indicate the number of cells that are in the topological (incidence or adjacency) relation with the given cell, as well as the amount and position of coordinate changes needed to obtain these cells. Increasing/decreasing the coordinate values of the given cell by the indicated amount produces exactly the cells in the corresponding topological relation. Tests on the coordinates of the given cell, whether they are equal to a value, are meant modulo 8.

For odd edges, the coordinate changes are the same as for even edges in Table 4, but odd/even are exchanged and coordinate value $\equiv 0$ in the edge plays the role of the value $\equiv 2$, while coordinate value $\equiv 4$ plays the role of the value $\equiv 6$ (mod 8).

Table 2: Topological data for voxels.

Relations for even / odd voxel	increments (multiset)
8 adjacent odd / even voxels	$\{\pm 6, \mp 2, \mp 2, \mp 2\}$
6 adjacent even / odd voxels	$\{+4, +4, -4, -4\}$
8 incident hexagonal faces	$\{\pm 3, \mp 1, \mp 1, \mp 1\}$
6 incident even / odd square faces	$\{+2, +2, -2, -2\}$
24 incident even / odd edges	$\{0, \pm 4, \mp 2, \mp 2\}$
12 incident odd / even edges	$\{0, 0, +2, -2\}$
24 incident vertices	$\{+1, -1, +3, -3\}$

Thus, the complete information on topological (incidence and adjacency) relations between the cells in the BCC grid can be obtained directly from cell coordinates through simple integer operations, without the

Table 3: Topological data for faces.

Relations for hexagonal face	increments (multiset)	
2 incident voxels	$\{\pm 3, \mp 1, \mp 1, \mp 1\}$	the different coord. has ± 3
6 adjacent hex-faces	$\{+2, +2, -2, -2\}$	
6 adjacent quad-faces	$\{\pm 3, \mp 1, \mp 1, \mp 1\}$	the different coord. has ∓ 1
6 incident edges	$\{+1, +1, -1, -1\}$	
6 incident vertices	$\{+2, -2, 0, 0\}$	the different coord. has 0

Relations for even / odd square face	increments (multiset)	
2 incident even / odd voxels	$\{+3, +3, -3, -3\}$	one pair of equal coords. has $+3, +3$, the other one $-3, -3$
8 adjacent hex-faces	$\{\pm 3, \mp 1, \mp 1, \mp 1\}$	
4 incident even / odd edges	$\{0, 0, +2, -2\}$	one pair of equal coords. has $0, 0$
4 incident vertices	$\{+1, +1, -1, -1\}$	each pair of equal coords. has $+1, -1$

use of pointers or other auxiliary data structures. The algorithms described in Section 7 use this data.

7 Algorithms and implementation

We implemented the BCC grid with the 4-valued coordinate system and topological relations. On top of it, we have developed a set of algorithms covering each stage of the following process:

- take as input a 3D binary image in the cubic grid and repair it, thus generating a well-composed image in the BCC grid;
- extract the boundary surface of black voxels in the well-composed BCC image;
- compute the Euler characteristics of the black voxels by computing the Euler characteristic of the boundary surface.

Besides the image repairing problem, considered in this paper, these algorithms also show how morphological tasks can be easily and efficiently supported in the BCC grid thanks to our coordinate system.

7.1 Image Repairing Algorithm

The repairing algorithm first loops on input cubes and sets the corresponding even voxels as black. Then, it loops on black even voxels and, for each of them, tests whether the eight adjacent odd voxels must be filled, by applying the rules defined in Section 4.1. Both the black/white status of the eight voxels adjacent to the odd voxel under examination, and the various configurations corresponding to non-manifold edges and vertices, are coded on 8-bit masks, and they are compared through bit-wise logical operators.

Figure 6 shows an example of an input cubical image (a) and the corresponding well-composed BCC image (b). Although homotopicaly equivalent, the BCC image is visually very different from the original one. For visualization purposes, it may be geometrically realized in an alternative way, as illustrated in Figure 6(d) and explained in the following. The quad-faces of even cubes are expanded and the ones of odd-cubes are reduced, until the former become octagons inscribed in the original square faces of the cubes, and

Table 4: Topological data for even edges.

Relations for even edge	increments (multiset)	
2 incident even voxels	$\{\pm 2, \pm 2, 0, \mp 4\}$	the two equal coords. have $\pm 2, \pm 2$; if they become 0 (4), then the coord. $= 0$ has 0 (∓ 4), the one $= 4$ has ∓ 4 (0)
1 incident odd voxel	$\{0, 0, +2, -2\}$	the two equal coords. have $0, 0$; if they are $= 2$ ($= 6$), then the coord. $= 0$ has $+2$ (-2), the one $= 4$ has -2 ($+2$)
2 incident hex-faces	$\{+1, +1, -1, -1\}$	the two equal coords. have equal increment
1 incident even quad-face	$\{0, 0, +2, -2\}$	the two equal coords. have $0, 0$; if they are $= 2$ ($= 6$), then the coord. $= 4$ has $+2$ (-2), the one $= 0$ has -2 ($+2$)
2 adjacent even edges	$\{+2, +2, -2, -2\}$	if the two equal coords. are $= 2$ (6), then the coord. $= 0$ has -2 ($+2$) and the one $= 4$ has $+2$ (-2)
4 adjacent odd edges	$\{0, 0, +2, -2\}$	exactly one of the two equal coords. has 0; if the other equal coord. has $+2$ (-2), then -2 ($+2$) is given to the coord. $= 0$ ($= 4$) if the two equal coords. where $= 2$; vice versa if they were $= 6$
2 incident vertices	$\{+1, +1, -1, -1\}$	the two equal coords. have $+1, -1$, if they were $= 2$ (6), then the coord. $= 0$ has -1 ($+1$) and the coord. $= 4$ has $+1$ (-1)

the latter collapse. Also, the common hex-faces of even and odd voxels collapse to triangles, and odd voxels reduce to diamonds located at the positions of vertices of the original cubes. This alternative realization is shown in Figure 6(c).

7.2 Extraction of Boundary Surface

Given the repaired BCC image, we reconstruct the boundary B between black and white voxels by applying an implementation, based on our coordinate system, of the classical algorithm. This is basically a breadth-first search (BFS) of the graph whose nodes are the boundary faces, and whose arcs correspond to their adjacency relation.

The algorithm starts from one given initial face f_0 and uses a growing paradigm supported by a queue. Each time a new boundary face f is found (the first one is f_0), the boundary faces adjacent to f are enqueued. The algorithm loops until the queue is empty. In order to avoid examining the same boundary face more times, faces are marked on insertion in the queue.

The boundary of an image may consist of more than one connected component, therefore the above described algorithm must be iterated with an initial face for each boundary surface. We add a preprocessing stage in which we perform a BFS visiting all voxels, and insert all exposed voxels in a queue. Then, we extract an exposed voxel V at a time and, if V has some boundary face f, which is not marked (i.e., f is not in an already reconstructed boundary surface), then we run boundary reconstruction from f.

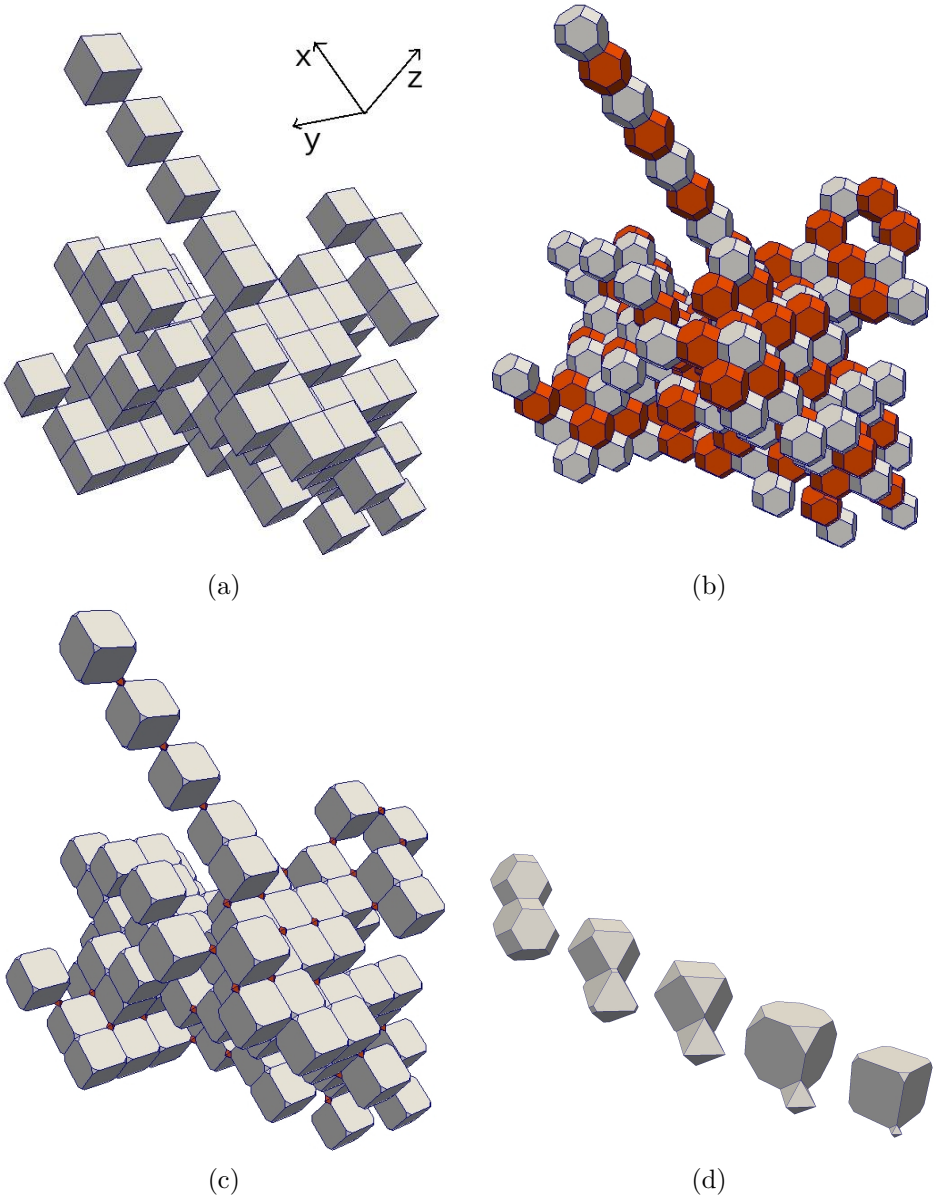


Figure 6: (a) An example of a randomly generated image with 100 cubes. (b) The repaired BCC image and (c) its alternative visualization mode. Even voxels, corresponding to original cubes, are white and added odd voxels are red. (d) Snapshots of the process of shrinking odd voxels and expanding even ones.

Table 5: Topological data for vertices.

Relations for vertex	increments (multiset)	
2 incident even voxels	$\{+1,-1,+3,-3\}$	the coords. $=1,7$ have opposite increment, and so the two $=3,5$
2 incident odd voxels	$\{+1,-1,+3,-3\}$	the two coords. $=1,3$ have opposite increment, and so the two $=5,7$
4 incident hex-faces	$\{0,0,+2,-2\}$	either the two coords. $=1,5$ or the ones $=3,7$ have $0,0$
1 incident even quad-face	$\{+1,+1,-1,-1\}$	the coords. $=1,5$ have $+1,+1$
1 incident odd quad-face	$\{+1,+1,-1,-1\}$	the coords. $=1,5$ have $-1,-1$
2 incident even edges	$\{+1,+1,-1,-1\}$	the coords. $=1,7$ have same increment, and so the ones $=3,5$
2 incident odd edges	$\{+1,+1,-1,-1\}$	the coords. $=1,3$ have same increment, and so the ones $=5,7$
4 adjacent vertices	$\{0,0,+2,-2\}$	either the two coords. $=1,3$ or the ones $=5,7$ have $0,0$; the smaller (mod 8) of the other two has $+2$

Thanks to our coordinate system, each cell is simply encoded as a tuple of four integers, and all cells incident and adjacent to it are retrieved in constant time. Marking or checking marks is done by means of matrices in constant time, as explained in Section 7.4. Thus, the time complexity of the algorithm is linear in the number of boundary faces.

7.3 Computation of the Euler Characteristic

The set of black voxels is a 3-manifold with boundary in $\mathbb{R}^3$. Its Euler characteristic is equal to one half of the Euler characteristic of the boundary surface B [Hat2001, Che2008]. Computation of Euler characteristic of B requires counting the number n_v of vertices, n_e of edges and n_f of faces in the boundary for applying the Euler formula:

$$\chi(B) = n_v - n_e + n_f$$

Given the computed faces of a boundary surface, we traverse the surface and count faces, edges and vertices. The algorithm performs a simple loop on faces. For each boundary face f, it counts the face f itself, and each edge and vertex of f. In order to avoid counting many times the same edge or vertex, we consider a total order of the faces, for example based on lexicographic order of 4-tuples (p,q,r,s). Then, an edge e is counted only if the current boundary face precedes the other boundary face incident in e, and a vertex v is counted only if the current boundary face is minimum among the boundary faces incident in v.

In the BCC grid, each face has a constant number of incident vertices (four or six) and of adjacent faces (double the number of vertices), and each vertex has a constant number of incident faces (six). Therefore, each iteration of the loop takes a constant time.

Note that we can merge the computation of Euler characteristics within the boundary extraction algorithm.

7.4 Marking voxels and faces

Many algorithms need to mark cells: either all cells or, more often, cells of a specific dimension (voxels, faces, edges, vertices). For example, the boundary tracking and Euler characteristic algorithms (see Sections 7.2 and 7.3) need to mark voxels and faces.

For voxels, a simple way to mark them, without waste of space, is by using two 3D matrices, one for even voxels and one for odd voxels, where the three indexes are the Cartesian coordinates of the center of

the voxel divided by 2. The Cartesian coordinates of the center of a voxel (p, q, r, s) are:
$x = (q + s)/4$
$y = (s + p)/4$
$z = (p + q)/4$
and they are even (odd) integers for even (odd) voxels. Our division by 2 takes into account the sign, i.e., for odd n we take $(n - 1)/2$. For example, even voxel $(4, -4, 4, -4)$ and odd voxel $(6, -2, -2, -2)$ are mapped to the same three indexes $-1, 0, 0$ in the two matrices.

Our implementation uses the same matrices for marking voxels and their faces. The smallest allocation unit is a character (8 bits), while Boolean voxel marks need just 1 bit. We use the other 7 bits for marking faces, in the following way. Each voxel has 14 faces, and each face has two incident voxels. For each voxel V we consider just the seven faces lying in the positive direction of axes x, y, z, p, q, r, s, and we say that V is the *reference voxel* for those faces. For the other seven faces of V, the reference voxel will be the other voxel incident in each of them. The same matrix element (character) stores the mark for a voxel and for the seven faces having that voxel as their reference voxel.

The algorithm uses the same matrix pair both for representing the input (i.e., for marking the black voxels) and for marking the faces during the algorithm.

The same approach can be used for marking edges and vertices of the BCC grid, if necessary. A voxel will be the reference voxel for 12 of its 36 incident edges (as each edge is shared by three voxels), and for six of its 24 incident vertices (as each vertex is common to four voxels). So for marking vertices and edges we need another 18 bits. This makes a total of 26 bits, which fit into a 32-bit integer.

7.5 Results

In order to test the algorithms, we developed a generator which creates a connected set of m cubes within an n^3 cubic grid, for given m and n. The generator fills the n cubes on the diagonal of the grid; then, starting from the central cube, it iteratively moves to a random direction, among the 26 possible ones, until $m - n$ other cubes have been filled.

We have implemented the algorithms in C language and tested them on a PC equipped with an Intel CPU i7-2600K CPU at 3.4 Gigahertz with 32 Gigabytes of RAM.

Input images have a number m of black cubes ranging from 100 to 1 million, contained in a grid of side $n = \sqrt{m}$. Table 6 shows statistics on input and output size, memory requirements and execution times. Numbers are computed on a basis of 20 executions, each time with a new randomly generated image. For faces, we show separately the total number over all connected components of the boundary surface, and in outer and inner boundaries, since the latter are much smaller. Input images have a single boundary surface, and our algorithm finds it first, because the first iteration of boundary reconstruction starts from an exposed face of the voxel with smallest coordinates. The table shows the number of through holes h of the surfaces, which is derived from Euler characteristic since, for a connected and closed surface, $\chi = 2 - 2h$.

We note that the number of BCC black voxels in the repaired image is about twice the number of input black cubes. This agrees with the consideration that, if black voxels in the cubic image were arranged to form one large cube, then $\simeq m$ odd voxels were filled (all the internal ones, plus the exposed ones on the three faces of the large cube lying in positive axis directions. All such odd voxels are added by rule (iii), since rules (i) and (ii) are for critical configurations. In another well-formed cubic image, fewer odd voxels would be added, because it has fewer black edges. For a generic cubic image, more that m odd voxels can be filled, and their number depends on the amount of critical configurations.

Of the alternative methods for repairing 3D binary images, one [Siq2008] does not guarantee the homotopy equivalence between the original and the repaired image. The other two [Gon2017, Ros1998] have such topological guarantee for the constructed (polyhedral or cubical) complex. However, [Ros1998] is clearly impractical for large images. So our direct competitor is [Gon2017].

This method relies on examining various 3D specific configurations of $2 \times 2 \times 2$ voxels, and the retrieval of topological relations between cells through structuring elements is not intuitive or easily implementable. The shape of polyhedra depend on the type of 11 different critical configurations, while we use only one

Table 6: Results of applying the algorithms to 3D images composed of an increasing number of black cubes. R stands for repairing, B stands for boundary tracking, E stands for computation of Euler characteristic. Statistics for outer and inner surfaces are shown separately. Times are in milliseconds, sizes are in bytes. Where two numbers are shown, they are minimum and maximum. K stands for 10^3 and M for 10^6. The last line shows the matrix sizes as printed by Matlab. For the two sets of large images, Matlab went out-of-memory.

image cubes	10^3 100	32^3 1000	100^3 10K	316^3 100K	1000^3 1M
voxels	180 – 197	1936 – 1991	$\simeq$ 20K	$\simeq$ 202K	$\simeq$ 2M
surfaces	1 – 2	4 – 14	218 – 272	2165 – 2429	18 – 19K
faces (tot)	1170 – 1554	14 – 18K	88 – 118K	527 – 568K	3.2 – 3.4M
faces (out)	1170 – 1554	14 – 16K	81 – 111K	527 – 568K	2.7 – 2.8M
faces (in)	14 – 26	14 – 70	14 – 322	14 – 774	14 – 1200
holes (out)	1 – 7	82 – 125	583 – 926	3603 – 4194	22 – 23K
holes (in)	0 – 1	0 – 1	0 – 1	0 – 3	0 – 4
R time	1	7 – 18	79 – 148	1354 – 1441	31 – 34K
B time	8 – 17	81 – 130	422 – 600	2349 – 2622	14 – 16K
E time	13 – 23	106 – 198	644 – 857	3308 – 3773	19 – 22K
queue bytes	2592 – 3936	10 – 15K	40k – 63k	91 – 117K	260 – 309K
queue faces	81 – 123	299 – 479	1236 – 1982	2838 – 3671	8110 – 9670
matrix bytes	4394	86K	2M	65M	2018M
matrix size	$2 \cdot 13^3$	$2 \cdot 35^3$	$2 \cdot 103^3$	$2 \cdot 319^3$	$2 \cdot 1003^3$
matrix size [Gon2017]	$2 \cdot 45^3$	$2 \cdot 133^3$	$2 \cdot 405^3$	–	–

type of polyhedra: the truncated octahedron of the BCC grid. Similarly to ours, it makes heavy use of a combinatorial coordinate system that is naturally defined for the cubical grid.

A Matlab implementation of [Gon2017] is available and we tried it on our test images. Matlab is a high-level scientific computation environment, so comparing execution times with a C program makes no sense, but still some considerations about memory requirements can be drawn. The last line of Table 6 shows the sizes of matrices generated by [Gon2017] on the same images.

The code computes a sequence of matrices, where the last one encodes the repaired image and its boundary. Given a cubic image of resolution n^3, there matrices have size $\simeq (4n)^3$. Since each matrix is used to compute the next one, we assume that an optimized implementation could keep just two matrices at a time in memory. Matrices contain Boolean values, so an optimized implementation could pack each 8 elements into a single byte, thus memory requirements would be $\simeq 2(4n)^3/8 = 16n^3$ bytes.

To mark black voxels and visited faces, our program uses two matrices of $\simeq n^3$ bytes each (as explained in Section 7.4), therefore requiring $\simeq 2n^3$ bytes, and this dominates memory requirements, as the queue size is smaller (see Table 6). Based on these considerations, we can say that [Gon2017] is more memory-consuming. Moreover, for our method, marking cells through matrices is an implementation choice. For sparse images (i.e., few black voxels compared to image resolution) we could rather keep just black voxels and visited faces into a search structure allowing linear size and sub-linear access time (such as a balanced search tree or a hash table), while the approach in [Gon2017] is inherently matrix-based.

8 Concluding Remarks

We propose an algorithm to locally repair a 3D binary image I by passing to the BCC grid. The repaired complex Γ is well-composed and homotopy equivalent to the corresponding cubical complex Q with 26-connectivity. Bressenham-like algorithms for ray-casting in the BCC grid have already been developed

[Iba1997, Iba1998], and can be coupled with our repairing algorithm for tunnel-free rendering of digitized surfaces.

We propose a 4-valued symmetric integer-valued combinatorial coordinate system that addresses all cells in the BCC grid and enables an easy and straightforward navigation through the grid. We demonstrate the usefulness of this system by developing algorithms for boundary tracking and Euler computation in the BCC grid, as examples of tasks relevant for visualization and topological analysis of objects. This work opens the way to the implementation in the BCC grid of algorithms that manipulate lower-dimensional cells and their topological relations, in particular to algorithms for (co)homology computation.

At the state of the art, our approach has just one competitor, that is the matrix-based approach developed by Gonzalez et al. [Gon2017], but this has larger memory requirements and is less intuitive. The produced polyhedral complex [Gon2017] might be visually more appealing than the complex Γ in the BCC grid, but we believe that the alternative geometrical realization of Γ balances this drawback.

The repaired complex Γ produced by our algorithm emulates the 26-connectivity for the black cubes. At present, we are working on a repairing algorithm that will emulate the 18-connectivity for the black cubes by passing to the face-centered cubic (FCC) grid.

Acknowledgment

This work has been partially supported by the Ministry of Education and Science of the Republic of Serbia within the Project No. 34014.

References

[Bou2015] Boutry, N., Géraud, T., Najman, L., 2015. How to make nD images well-composed without interpolation. In: 2015 IEEE International Conference on Image Processing, ICIP 2015. pp. 2149–2153.

[Bri2009] Brimkov, V. E., 2009. Formulas for the number of $(n-2)$-gaps of binary objects in arbitrary dimension. Discrete Applied Mathematics 157 (3), 452–463.

[Bri2006] Brimkov, V. E., Maimone, A., Nordo, G., 2006. Counting Gaps in Binary Pictures. In: Combinatorial Image Analysis, 11th International Workshop, IWCIA. pp. 16–24.

[Bri2005] Brimkov, V. E., Maimone, A., Nordo, G., Barneva, R. P., Klette, R., 2005. The Number of Gaps in Binary Pictures. In: Advances in Visual Computing, First International Symposium, ISVC. pp. 35–42.

[Che2008] Chen, L., Rong, Y., 2008. Linear time recognition algorithms for topological invariants in 3D. In: 19th International Conference on Pattern Recognition (ICPR. pp. 1–4.

[Coh1997] Cohen-Or, D., Kaufman, A. E., 1997. 3D Line Voxelization and Connectivity Control. IEEE Computer Graphics and Applications 17 (6), 80–87.

[Com2016] Čomić, L., Nagy, B., 2016. A combinatorial coordinate system for the body-centered cubic grid. Graphical Models 87, 11–22.

[Gon2015] González-Díaz, R., Jiménez, M. J., Medrano, B., 2015. 3D well-composed polyhedral complexes. Discrete Applied Mathematics 183, 59–77.

[Gon2017] González-Díaz, R., Jiménez, M. J., Medrano, B., 2017. Efficiently Storing Well-Composed Polyhedral Complexes Computed Over 3D Binary Images. Journal of Mathematical Imaging and Vision 59 (1), 106–122.

[Hat2001] Hatcher, A., 2001. Algebraic Topology. Cambridge University Press.

[Iba1997] Ibáñez, L., Hamitouche, C., Roux, C., 1997. Ray-tracing and 3D Objects Representation in the BCC and FCC Grids. In: Discrete Geometry for Computer Imagery, 7th International Workshop, (DGCI). pp. 235–242.

[Iba1998] Ibáñez, L., Hamitouche, C., Roux, C., 1998. Ray casting in the BCC grid applied to 3D medical image visualization. In: Proceedings of the 20th Annual International Conference of the IEEE Engineering in Medicine and Biology Society. Vol.20 Biomedical Engineering Towards the Year 2000 and Beyond (Cat. No.98CH36286). Vol. 2. pp. 548–551 vol.2.

[Kit2004] Kittel, C., 2004. Introduction to solid state physics. Wiley, New York.

[Kle2004] Klette, R., Rosenfeld, A., 2004. Digital geometry. Geometric methods for digital picture analysis. Morgan Kaufmann Publishers, San Francisco, Amsterdam.

[Kon1989] Kong, T. Y., Rosenfeld, A., 1989. Digital topology: Introduction and survey. Computer Vision, Graphics, and Image Processing 48 (3), 357–393.

[Kov2008] Kovalevsky, V. A., 2008. Geometry of Locally Finite Spaces (Computer Agreeable Topology and Algorithms for Computer Imagery). Editing House Dr. Bärbel Kovalevski, Berlin.

[Lac2000] Lachaud, J., Montanvert, A., 2000. Continuous Analogs of Digital Boundaries: A Topological Approach to Iso-Surfaces. Graphical Models 62 (3), 129–164.

[Lat1997] Latecki, L. J., 1997. 3D Well-Composed Pictures. CVGIP: Graphical Model and Image Processing 59 (3), 164–172.

[Mai2011] Maimone, A., Nordo, G., 2011. On 1-gaps in 3d digital objects. Filomat 22 (3), 85–91.

[Mai2013] Maimone, A., Nordo, G., 2013. A formula for the number of $(n-2)$-gaps in digital n-objects. Filomat 27 (4), 547–557.

[Mor1980] Morgenthaler, D., 1980. Three-Dimensional Digital Topology: the Genus. Tech. Rep. TR-980, University of Maryland, College Park, MD 20742.

[Nag2008] Nagy, B., Strand, R., 2008. Non-traditional grids embedded in Z^{n}. International Journal of Shape Modeling 14 (2), 209–228.

[Ros1998] Rosenfeld, A., Kong, T. Y., Nakamura, A., 1998. Topology-Preserving Deformations of Two-Valued Digital Pictures. Graphical Models and Image Processing 60 (1), 24–34.

[Siq2008] Siqueira, M., Latecki, L. J., Tustison, N. J., Gallier, J. H., Gee, J. C., 2008. Topological Repairing of 3D Digital Images. Journal of Mathematical Imaging and Vision 30 (3), 249–274.

A Pseudocode of algorithms

The pseudocode of the image repairing algorithm and of its auxiliary test function is in Table 7.

The pseudocode of the algorithm which, given an initial face f_0, finds a single connected component of the boundary surface, is shown in Table 8.

The pseudocode of the algorithm for computing the Euler characteristic is shown in Table 9. In each iteration of the `for` loop, there are two faces adjacent to the current face f across each of its edges, and exactly one of these two faces belongs to the boundary surface. In our implementation, the topological relations provide sorted and aligned lists of vertices, edges and faces for a face f, according to the following conventions. The k incident vertices and edges ($k = 4$ or 6) are sorted around the boundary of f, and the endpoints of the i-th edge are vertices number i and $i+1$. Adjacent faces are divided in two sections, each sorted around f, where faces in position i and $i+k$ are both adjacent to f along the the i-th incident edge of f. So, in Table 9, f_{curr} is the boundary face adjacent to the i-th edge of f, and $av[i]$ is one of the endpoints of such an edge.

Table 7: Pseudocode of the repairing algorithm and the auxiliary function testing whether a vertex must be colored.

```
algorithm Repair ()
 0   set all voxels as white;
     // set black even voxels:
 1   for each input cube (x, y, z)
 2       p=4*(-x+y+z);
 3       q=4*(x-y+z);
 4       r= 4*(-x-y-z);
 5       s= 4*(x+y-z);
 6       set voxel (p, q, r, s) as black;
 7   end for
     // set black odd voxels
 8   for each even black voxel b
 9       for each 8-adjacent odd voxel v of b
10           if (v is not black AND MustBeFilled(v))
11               set voxel v as black;
12       end for
13   end for
```

```
boolean function MustBeFilled (cell v)
     // shortcuts for easy configurations:
 0   if (v has 8 incident black voxels) return true;
 1   if (v has 1 incident black voxel) return false;
     // rule (iii):
 2   for each direction d ∈ {−x, −y, −z}
 3           if (incident edge in direction d is black) return true;
     // rule (ii):
 4   for each direction d ∈ {+x, +y, +z}
 5           if (incident edge in direction d is white)
 6                   return (incident edge in direction −d is non-manifold);
     // rule (i):
 7   if (v has some incident white edge) return false;
 8   if (v is non-manifold) return true;
 9   if (v has some non-manifold incident edge) return true;
10   return false;
```

Table 8: Pseudocode of the boundary tracking algorithm. Input parameter is one face f_0 belonging to the boundary surface.

```
algorithm FindBoundary (cell f_0)
 0  set all faces as not marked;
 1  Q = new empty queue;
 2  add f_0 to queue Q;
 3  mark f_0;
 4  while (queue Q is not empty)
 5      f = extract first face from queue Q;
 6      output f;
 7      a[0...11] = get adjacent faces of f;
 8      for (int i=0; i < number of adjacent faces; i++)
 9          if (a[i] is not marked)
10              b[0...1] = incident voxels of a[i];
11              if (b[0] and b[1] are one black and one white)
12                  add a[i] to queue Q;
13                  mark a[i];
14      end for
15  end while
```

Table 9: Pseudocode of the algorithm for counting the number n_f of faces, n_e of edges, and n_v of vertices.

```
algorithm EulerCharacteristic (cell set F)
 0  init n_f, n_e, n_v = 0;
 1  for each face f in F
 2      av[0...5] = get incident vertices of f;
 3      af[0...11] = get adjacent faces of f;
 4      n = number of vertices of f;
 5      for (int i=0; i < n; i++)
 6          f_curr = the boundary face between af[i] and af[i + n];
 7          if (f < f_curr)
 8              increment n_e;
 9              vf[0...5] = get incident faces of av[i];
10              flag = true;
11              for (int j=0; flag AND j < 6; j++)
12                  if (vf[j] < f) flag = false;
13              end for
14              if (flag) increment n_v;
15      increment n_f;
16      end for
17  end for
```