Efficient algorithms for boundary extraction of 2D and 3D orthogonal pseudomanifolds

Marc Vigo, Núria Pla, Dolors Ayala, Jonàs Martínez

Dept. de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Edifici ETSEIB, Diagonal 647, 8a planta
E - 08028 Barcelona, SPAIN

Abstract

In this paper we present algorithms to extract the boundary representation of orthogonal polygons and polyhedra, either manifold or pseudomanifold. The algorithms we develop reconstruct not only the polygons of the boundaries but also the hole-face inclusion relationship. Our algorithms have a simple input so they can be used to convert many different kinds of models to B-Rep. In the 2D case, the input is the set of vertices, and in the 3D case, some small additional information must be supplied for every vertex. All proposed algorithms run in $\mathcal{O}(n \log n)$ time and use $\mathcal{O}(n)$ space, where n is the number of vertices of the input. Moreover, we explain how to use our proposal to extract the boundary from the well-known voxel and octree models as well as from three vertex-based models found in the related literature: the neighbourhood, the EVM, and the weighted vertex list models.

1. Introduction

Extracting the boundary of a 2D or a 3D binary image is a fundamental operation in image processing. In other fields as NC (Numerical control) data generation, obtaining the cutting areas of sculptured surfaces can be performed by representing the surface first using a regular grid model which is equivalent to a 2D binary image.

Boundary extraction of images is approached by two main methods: polygonal and digital. Polygonal (or bevelled-form) methods represent the boundary as a set of edges (2D) or triangles (3D) [1]. Digital (or block-form) methods represent the boundary as a set of axis-aligned edges (2D) or faces (3D) that can be voxel size [2] or maximal, i.e., not restricted to any size [3].

Digital models exhibit formal properties such as closure, orientedness and connectedness whereas polygonal models and related techniques are mainly devoted to visualisation purposes [4]. Polygonal models tend to produce poorly shaped and degenerate polygons that can be partially corrected with a smoothing post-process [5] or by modifying the grid [6].

A common drawback of the existing techniques is that the resulting contours consist of several little edges (2D) or little quadrangular or triangular faces (3D). Several adaptive attempts have been developed to reduce this redundancy. A kd-tree can be used to extract a manifold quadrilateral mesh [7] while the dual contouring method uses an octree and produces crack-free manifold contours [8].

Alternatively, digital methods interpret the boundary of a binary image as orthogonal polygons or polyhedra that can be manifold or pseudomanifold. An adjacency pair (m, n) is associated to a binary image, meaning that foreground voxels are m-adjacent and background voxels are n-adjacent [9]. Using the same adjacency relations for the foreground and background voxels involves that paradoxes can be avoided by restricting these pairs to (4,8) and (8,4) for 2D, and (6,26) and (26,6) for 3D [9]. However images with this adjacency pair present pseudomanifold configurations.

In this work we present a method that extracts a B-Rep from orthogonal polygons and polyhedra obtaining all of its oriented contours and with the corresponding inclusion relationships. The algorithms presented run in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space, where n is the number of vertices, and can deal with manifolds as well as pseudomanifolds. The input of the presented methods is the set of vertices (2D, 3D) and some neighbourhood information (3D). We also present methods to obtain this input information from several other models: voxel and octree models, neighbourhood representation, Extreme Vertices Model (EVM) and weighted vertex list representation.

The constrained structure of orthogonal polyhedra has enabled advances on complex problems for arbitrary polyhedra, for example, unfolding the surface of a polyhedron to a polygonal net [10] or computing skeletons of polyhedra [11].

2. Previous work

Several research attempt to extract orthogonal boundaries from other models and also seek for suitable representations for orthogonal objects. Some approaches are based on spatial enumeration models, while others use vertex lists to represent orthogonal polyhedra.

Regarding spatial enumeration models, Montani and Scopigno [12] proposed algorithms to convert quadtrees to polygons and octrees to polyhedra. They first convert the hierarchical structure into a strip-based model (the Parallel Connected Stripes, PCS), and then the boundary is reconstructed from PCS. The complexity of this proposal depends on the number of stripes in the PCS, which can be very high in the worst case. The authors do not mention how to treat non-manifolds, neither in 2D nor in 3D, and they avoid to compute inclusion relationships as they only consider the case with only one external contour. Karunakaran and Shringi [13] also developed a program that converts an octree to a B-Rep. The main drawback of their proposal is that it is very specific for NC machines, and therefore it is difficult to generalise as an algorithm that admits other input models and that outputs a full B-Rep.

Heijmans [14] computes the zonal graph for 2D images and uses it to perform several 2D image processes. The zonal graph is a graph representing the inclusion relationships between polygons in such a way that every vertex of the graph represents a polygon, being specified also whether it belongs to the foreground (face polygon) or to the background (hole polygon). Park and Choi [15] present a different approach on boundary extraction of a 2D image with all its polygons and inclusion relationships. The problem they solve is the extraction of cutting areas from a sculptured surface in the NC-data generation field. The sculptured surface is first discretised by sampling its z-values into a regular grid (a Z-map) that is then transformed into a 2D binary image and, from it, cutting areas are obtained. They use a run-length codification of the 2D image and devise an algorithm which is $\mathcal{O}(nr)$, nr being the number of runs. The boundary consists of axis-aligned polygons with maximal horizontal edges and vertical edges of pixel size, which is a consequence of the run-length codification used. Moreover, this method does not deal with pseudomanifold images. Concerning 3D images, Damiand [16] presents a topological map describing intervoxel boundaries and an incremental algorithm to extract the boundary. The model is a combinatorial map, and the method obtains first the B-Rep of every voxel and then applies removal operators for faces, edges and vertices.

The problem of inclusion relationship has also been studied for 3D shells. Gargantini et al. [17] presented a method that computes inclusion relationships between 3D shells and represent them in a region containment tree structure. Contours are given as border voxels and they use an octree to represent them. Then, they obtain the region containment tree by a traversal of the octree that labels connected components.

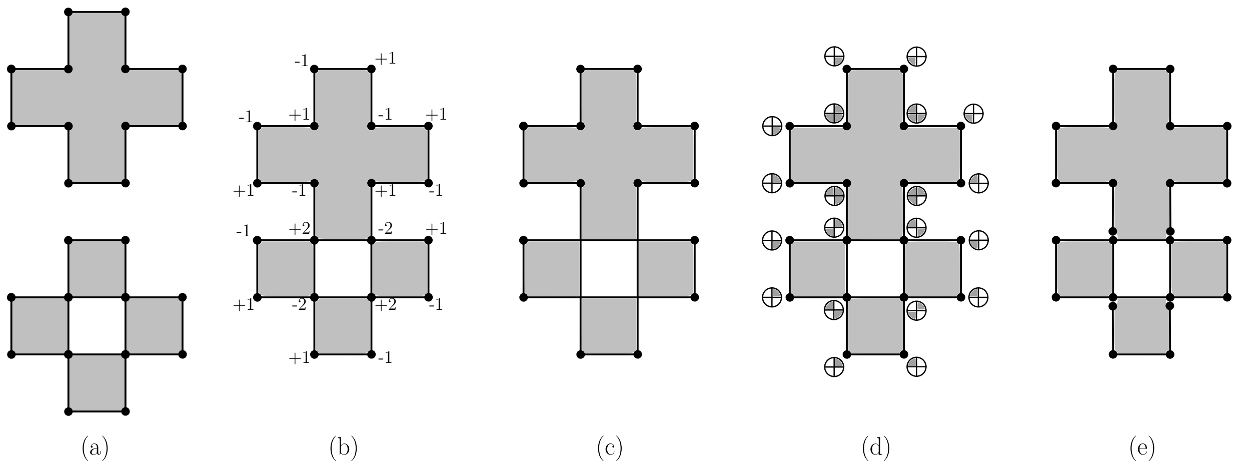


Figure 1: (a) Two 2D orthogonal pseudomanifolds sharing the same vertex set. (b)-(d) A pseudomanifold object represented using the weighted vertex set, the EVM and the neighbourhood models, respectively. (e) For the same object, the set of vertices that our approach needs as input.

A direct way to represent orthogonal polyhedra that results in a concise representation of binary images is by means of a vertex list. It has been shown that a list of all vertices without any additional information is ambiguous, even in 2D [18]. Figure 1a shows two different objects with the same vertices. However, vertex lists, with some additional information associated to the vertices, constitute an implicit and complete (non-ambiguous) representation model. Vertex lists represent the same kind of objects that can be represented by spatial enumeration models with the advantage that they require less storage and are not sensitive to translation [19]. With respect to run-length encoding, the number of runs depends on the precision of the underlying pixel (or voxel) model as well as on the run-length direction chosen. Although it is a compressed model, it represents the interior of the object while vertex lists represent the boundary. For example, the object in Figure 2a, has 78 vertices, 69 vertical runs and 62 horizontal runs considering the size of the shortest edge as the pixel size. However, the special case example of a $1 \times N$ size rectangle would have 4 vertices, 1 run in one direction and N runs in the other one.

In 2D, the problem of reconstructing a single orthogonal polygon from its set of vertices has been solved by O'Rourke [20]. The authors presented a $\mathcal{O}(n \log n)$ algorithm based on sorting points on any orthogonal line and then adding an edge between every other pair of points. This proposal is only valid for 2D manifolds and does not recover the hole-face containment relationship.

Esperança and Samet present and formalise a vertex list based model [21, 19], suitable for polygons and polyhedra as well as for multi-valued fields. To avoid confusion with other vertex list/set models, in the rest of the paper we refer to this model as the *weighted* vertex list representation. In this model, orthogonal polyhedra are represented by a subset of vertices - lexicographically sorted - with an associated weight needed for containment purposes. The model as well as the processes applied to it are recursive in the dimension, i. e., vertex representations in nD space can be represented and processed by vertex representations in $(n-1)D$ space using the project and unproject operators. These authors present methods to apply geometric transformations, set operations and display solids represented as vertex lists. Yet, they do not present a conversion algorithm from vertex lists to a more explicit B-Rep model.

Two other models that have been presented use lexicographically ordered vertex lists and are recursive in the dimension: the neighbourhood representation and the Extreme Vertices Model (EVM). The neighbourhood representation [18] is a model for orthogonal polyhedra used as an approximate model for reachable states in dynamical systems. It consists of a list of all polyhedron vertices that carry an associated *colour* for orientation purposes. The authors of this work present algorithms for point membership and Boolean operations. They also describe a method to detect facets of the polyhedron. The method has linear time complexity on the number of vertices. Still, it only finds the set of vertices associated to an orthogonal plane and it does not obtain an explicit B-Rep model as contours and hole-face inclusion relationships are not computed.

The EVM [22, 3] is another vertex list model that stores a subset of vertices, called extreme vertices, without any associated information, and represents indistinctly manifolds and pseudo-manifolds. General set membership including Boolean operations have been developed with this model as well as an algorithm [3] that extracts from the EVM an explicit B-Rep. The method obtains geometric information of oriented faces and all vertex coordinates with a sweep-plane based process, but to obtain inclusion relationship applies a quadratic complexity brute-force method consisting on point-polygon inclusion tests. Moreover, the method presents a flaw when dealing with vertices with 4 faces.

There is another model that uses a vertex list [23], but it is restricted to orthogonally convex polyhedra which are the subset of orthogonal polyhedra, which satisfy the following condition: any axis-parallel line intersects the polyhedron in at most one line segment. This restricted class of polyhedra can be represented with all its vertices and without any associated information. The authors present a $\mathcal{O}(n \log n)$ algorithm to extract faces from the vertex list, but they do not study the hole-face inclusion problem.

The input of the method presented in this paper is also a vertex list, with all the vertices and with an additional information associated to the vertices for orientation purposes. The presented method computes an explicit B-Rep with a sweep-plane based process, obtaining oriented faces with its contours and hole-face inclusion relationship. Moreover it solves the mentioned flaw concerning vertices with 4 faces. Remark that, in this work we do not address the problem of classifying 3D shells. The presented method can deal with orthogonal polyhedra with any number of external shells and cavities, and the output B-Rep has the relations face:polygons and polygon:vertices.

Figure 1 shows a 2D example represented with the weighted vertex list representation (b), the EVM (c) and the neighbourhood representation (d). See Section 8 and Appendix B for more details concerning these representations. Figure 1e shows this example represented as a suitable input for the approach presented in this paper: in 2D, non-manifold vertices must be repeated.

3. Notations and overview of the algorithms

In the 2D case, an edge with endpoints a and b is notated as ab. An edge ab is called *vertical* if it is parallel to the y axis, and *horizontal* if parallel to the x axis. A *polygon*, see [23], is a set P in a plane whose boundary is composed by an external closed curve and a set, possibly empty, of internal closed curves, defining the *holes* of the polygon. Each curve is the union of a finite number of line segments. The line segments are called *edges*. In this case, boundary closed curves are also called polygons, therefore, from now on we will also use the term polygon to reference a boundary curve. *Orthogonal polygons* are composed of boundary edges which are either vertical or horizontal.

An *orthogonal manifold polygon* is an orthogonal polygon such that two different edges intersect if and only if they are consecutive edges of one curve. A set of vertices V define an orthogonal manifold if and only if at each vertex of V meet exactly two edges, one vertical and the other horizontal (see Figure 2, left).

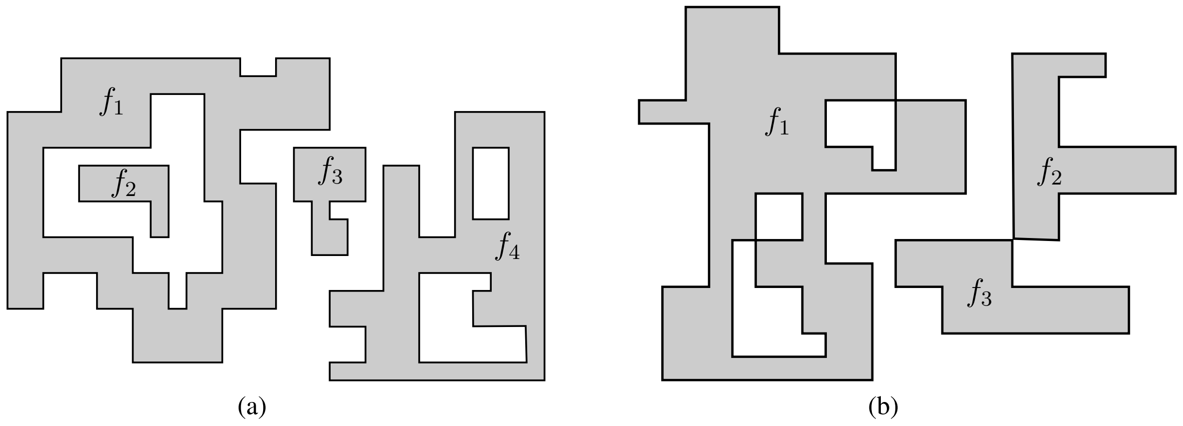


Figure 2: (a) An orthogonal manifold with 4 faces; (b) an orthogonal pseudomanifold.

A generalisation of an orthogonal manifold is an *orthogonal pseudomanifold*, with some vertices where four different edges meet (see Figure 2, right). In this case, also different edges are disjoint or intersect in a common vertex. Non-manifold edges can belong to the same face or to two different faces.

A *polyhedron* can be defined as a bounded three dimensional point set, which is equal to the closure of its interior points and whose boundary is contained in a finite union of planes [24]. If the boundary planes are orthogonal (that is, planes whose normal is parallel to one of the coordinate axis), the polyhedron is orthogonal.

An *orthogonal manifold polyhedron* can be defined as an orthogonal polyhedron such that each edge bounds exactly two faces and the union of all the faces and edges incident on a vertex defines a single cone with the vertex as the apex. In fact, the boundary of the polyhedron is a 2D manifold.

An *orthogonal pseudomanifold polyhedron* is an extension of the previous concept of polyhedron with non-manifold edges or vertices. A non-manifold edge is adjacent to four faces and a non-manifold vertex is a vertex with more than one cone of faces around it.

B-Rep models a solid indirectly by a representation of its boundary surface, which is a set of faces and topological information defining the relationships between the faces. The bounding curves of the faces are composed by straight lines defined by their vertices. Thus, faces can be represented as polygons, each polygon can be defined by a set of simple curves, which are defined by a set of vertices. The topological information says how vertices are connected to define each oriented curve. Inclusion relationship between curves, corresponding to the same face, is needed to know the holes of each face.

In particular, the output of our algorithm is a B-Rep model is composed by the set of vertices, the set of polygons, the set of faces and the relations face:polygons and polygon:vertices. The relation face:polygon distinguishes the external curve of the face from the rest of curves, the holes of the face. The relation polygon:vertices is stored as a connected list of vertices such that the normal of the face points toward the exterior of the solid. This implies that while the external polygon of a face is oriented clockwise, holes are oriented counterclockwise.

Notice that in this B-Rep model edges are implicit, but they can be easily recovered. Besides, it is also straightforward to obtain all other topological relations from the given ones. Pseudomanifolds can also be represented using this B-Rep. In this case, non-manifold vertices will have more than one cone of faces around them.

The algorithms implied in the extraction of the boundaries will be disclosed as follows. Initially, we propose a method to deal with manifold polygons, and then we reveal how this algorithm can be modified so that it handles non-manifold vertices. Next, we show how this 2D algorithm can be used to compute the B-Rep of orthogonal polyhedra, first treating the manifold case and then dealing with pseudomanifolds. Finally, we explain how to extract boundaries starting from different models.

One of our goals when developing the algorithms for boundary extraction is that the input should be as simple as possible, so that it can be easily applied to different kinds of models. In fact, for the 2D case the input is the set of all the vertices, some of them duplicated, and in the 3D case it is a set of vertices with some other simple geometrical information associated to them, as it will be explained.

4. Boundary extraction of a 2D manifold

The input of the 2D algorithm is the set of vertices V of an orthogonal manifold. The output is the boundary description of this manifold, i.e. a set of polygons, classified as external or holes. The output has also to include the hole:face relation information, that is to say, the set of holes corresponding to each external polygon.

4.1. Design of the 2D algorithm

The 2D algorithm is based on the following lemma, which is the same observed by O'Rourke [20] but stated in terms of a sorted list:

Lemma 1. *Let C be the set of faces of an orthogonal manifold in the plane, and let $Lxy = \{v_i\}, i = 1 \dots n$ be the lexicographically xy-sorted list of vertices of C. Then, the pairs (v_i, v_{i+1}), with $i = 1, 3, 5 \dots n-1$, of consecutive vertices in Lxy are the vertical edges of C.*

In the same way, the lexicographically yx-sorted list of vertices Lyx taken as consecutive pairs is the set of horizontal edges of C. Note that given an endpoint $v \in V$ of a vertical edge, one may find the other endpoint of the edge searching for v in Lxy and taking either the next element or the previous element, depending on if wether v is in an odd or even position in Lxy.

Therefore, to form a single polygon we may start from one of its vertical edges vw, search for it in Lxy, take the endpoint (v or w) that is in an even position, search for it in Lyx, take the corresponding endpoint, and so forth until vw is found again. If during this alternate search we mark the edges – pairs of vertices – visited, we could form all the polygons by traversing vertices of Lxy and forming a polygon each time a non-marked vertex is visited. This approach is very similar to the O'Rourke algorithm used to reconstruct polygons [20]. Notice that as stated this alternate search always forms polygons clockwise oriented, regardless of whether they are external or not. [1]

However, this approach does not recover the information on whether polygons are external or holes. Having all this in mind, our proposal is to perform a sweep line algorithm along the horizontal direction that processes vertical edges. Each time a vertical edge of a non-reconstructed polygon is found, we use the alternate search to form it. Besides, we keep track of the intersection between the vertical scan line and the manifold and we use it to recover the hole-face containment relationships during the sweep. The orientation of the polygons that are holes is changed in a post-process, once the sweep is finished. Figure 3 shows the sweep process.

Two elements characterise the sweep line schema [25]:

- The status data structure, S. In our case, S is the intersection of the vertical sweep line with the face set. It is stored as an y-ordered set of vertical segments, each one labelled with the identifier of the face it belongs to.
- The events (or stopping points). In our case, the events are the vertical edges, i.e. the elements in Lxy taken as consecutive pairs.

[1] Incidentally, if by contrast we start searching for the first edge in Lyx, the polygons are formed counter-clockwise.

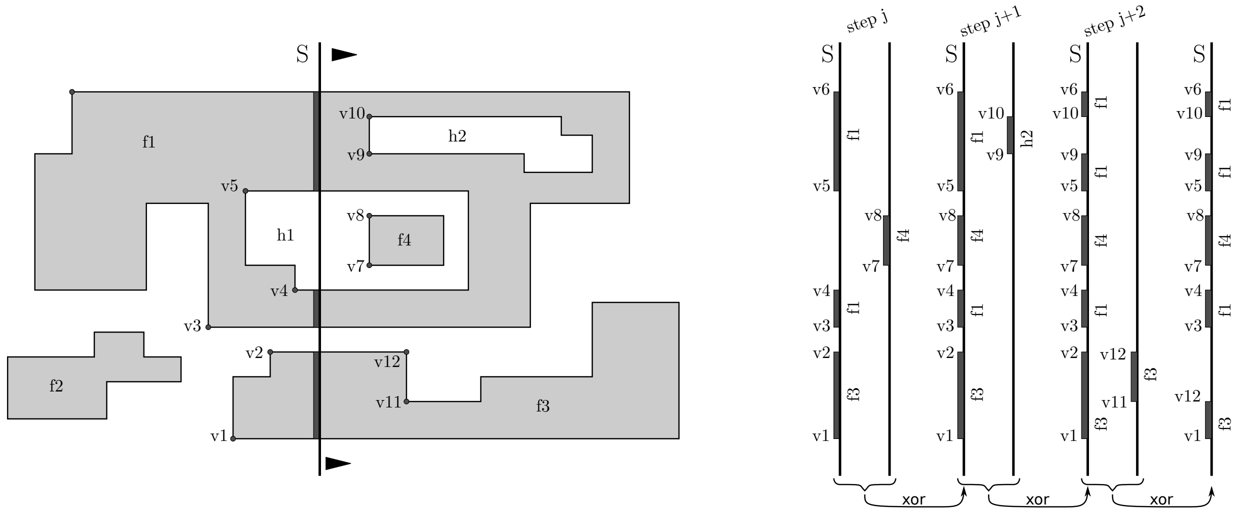


Figure 3: Sweep of a 2D set of faces. *Left*, an intermediate step j. *Right*, three steps showing how the section is updated using an xor operation with the new segments.

The invariant is that at any time all polygons with some vertices located left to the sweeping line have already been formed and classified, either as external or as holes of an external polygon also already formed.

In addition, the algorithm needs to know which face belongs to each visited vertical edge (the edge:face relationship). This information is stored as edge marks, i.e. edges are marked with the identifier of (or a pointer to) the face they belong to.

Processing an event – a vertical edge – is done in the following way: if the edge is already marked, only S needs to be updated, since it is an edge of an already formed polygon. Otherwise, if the edge is not marked, it must be the leftmost vertical edge of a polygon not formed. We update S and, with the help of S, we classify the edge as belonging to the external polygon of a new face or as a hole of an old face. Then, we form the polygon using the alternate search method, and finally we mark all the edges of the polygon recently formed as related to the face.

It is not difficult to see that updating S with a new vertical edge e can be done as an xor operation between S and the new segment (see Figure 3). This implies searching for the y coordinate of the lower endpoint of e in S and then merging the segment endpoints using its y values. Notice that at most three segments (the new one and two in S) are implied in this merge.

A non-marked edge is classified as belonging to an external polygon if it falls outside all segments in S. Otherwise, the edge must be part of a hole, and the edge falls strictly inside one of the intersection segments (in other words, both y values of the edge endpoints fall inside a segment of S). In fact, the labels of segments in S are stored in order to know the face that a hole belongs to.

As an example, Figure 3 shows an intermediate step j of the algorithm, and three updates of the section S. At the current state, S is composed of three segments (shaded intervals in the figure): one for face $f3$, and two for face $f1$. Polygons with some vertices on the left of the sweep line, i.e. external polygons $f1$, $f2$, $f3$ and hole $h1$, have already been formed and classified. Next three events are edges $v7v8$, $v9v10$ and $v11v12$. On the right of the figure, the update of S according to these 3 steps is shown. Notice that each update requires an xor operation between a set of ordered segments and a new segment. Processing edge $v7v8$ will activate the reconstruction of polygon $f4$, since it is a non-marked edge. This polygon will be classified as the external polygon of a face, since $v7v8$ falls outside segments in S, thus a the new face $f4$ is added to the set of faces. Then, S will be updated, adding a new face segment $f4$ to it between the two segments of face $f1$. When edge $v9v10$ is processed, polygon $h2$ will be formed, and it will be classified as a hole of $f1$ since $v9v10$ falls inside the upper segment of S. S will be updated by splitting the last segment into two new ones belonging to $f1$. When edge $v11v12$ is processed, as it is a marked edge (belonging to $f3$), no polygon will be formed, only S will be updated.

In Appendix A a Python function that implements the sweep for extracting the boundary of a 2D manifold is given and analysed.

4.2. Algorithm complexity

The following lemma states that the 2D algorithm has linearithmic worst-case time complexity on the number of input vertices.

Lemma 2. *The 2D algorithm for boundary extraction of an orthogonal set of faces has time complexity $\mathcal{O}(n \log n)$, being n the number of vertices of the input set.*

Proof. Lexicographical sorting of the set of vertices requires $\mathcal{O}(n \log n)$ time.

Once sorted, looking for a vertex when forming the polygons can be done with a binary search, and since each vertex is searched only once in Lxy or Lyx, forming all polygons requires $\mathcal{O}(n \log\ n)$ time.

Segments in S are ordered by their vertical position, therefore it may be implemented as a balanced binary search tree. Updating S with a new segment (the xor operation) requires first a search for one of the edge endpoints in S, and then a modification where at most three vertical segments are implied, so it requires a maximum of two insertions and/or two deletions in the binary tree. Search, insert and delete operations in a balanced binary search tree require $\mathcal{O}(\log\ n)$ time. Overall, n updates in S require $\mathcal{O}(n \log n)$ time.

Therefore, runtime for the 2D boundary extraction algorithm is $\mathcal{O}(n \log n)$. □

Space complexity of the 2D algorithm is $\mathcal{O}(n)$, because it requires several lists of vertices and a search tree whose worst-case size is $\mathcal{O}(n)$. The edge marks may be stored as inverse pointers either inside Lxy elements or in a parallel list.

5. Managing 2D pseudomanifolds

Note that the boundary of a face of a given manifold polyhedron needs not to be a manifold, thus 2D pseudomanifolds appear naturally from manifold polyhedra.

2D pseudomanifolds include non-manifold vertices, with four incident edges. Dealing with these vertices implies to properly form polygons following the pseudomanifold definition, so edge cycles have to separate the four edges properly. Following and taking into account that, later, when dealing with 3D models, the output polygons will define the faces of a polyhedron, the separation is done in such a way that the interior of each face must be a 2D connected component (see Figure 4).

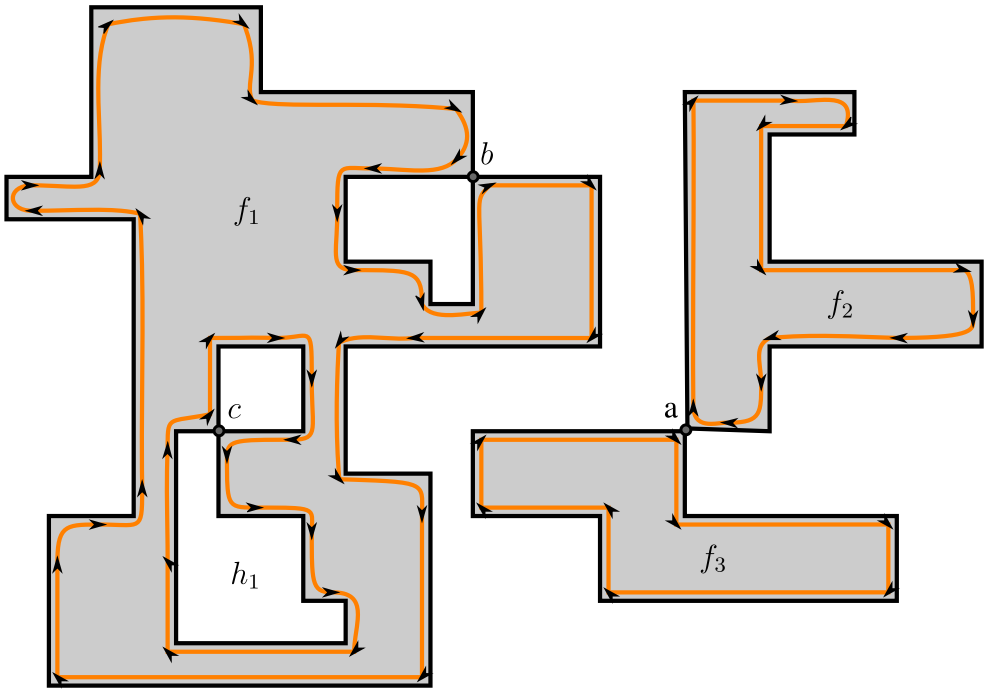


Figure 4: Polygon reconstruction for 2D pseudomanifolds. Three faces will be formed, one of them with a hole. The path shows the way polygons are formed.

As previously stated, in general a 2D pseudomanifold cannot be reconstructed from only the set of its vertices [18]. However, the method proposed in the previous section may be modified to handle pseudomanifolds by slightly varying the input and some parts of the algorithm. In fact, in our algorithm the input in the pseudomanifold polygon case is the set of vertices with duplicities for some of them. First of all, non-manifold vertices are inserted twice in the input list. If we proceed in this way, when the lexicographically sorted lists are computed, Lemma 1 still holds true: consecutive pairs of vertices in lists Lxy (respectively in Lyx) are the vertical (horizontal) edges of the manifold. Deciding if a vertex is manifold or non-manifold must be done with the help of the original representation scheme. For example, if the original model is a binary image, all pixel vertices with a non-manifold configuration around them must be inserted twice in the input list.

Second, we need to adapt the way polygons are formed so that non-manifold vertices may be managed. During the alternate search, when a duplicated vertex is found – a vertex identical to the next one in Lxy (or in Lyx) –, we have to choose between the two vertical or horizontal edges that emanate from it. Recall that all polygons are formed clockwise. So, if the polygon we are forming is an external one, we have to choose the edge that forms a convex vertex with the last edge. Otherwise, we could merge two non-connected components (vertex a in Figure 4) or form a polygon that leaves a complete cycle to its right, that would be incorrectly identified as one of its holes (vertex b in Figure 4). Therefore, in this case the edge that forms a convex vertex must be chosen. However, when a hole is being formed, we have to use the opposite rule: choose the edge that forms a concave vertex in the hole polygon (vertex c in Figure 4). This way to form polygons is consistent with a (4,8) underlying binary image. We could proceed in an alternate way for (8,4) images, choosing the edge that forms a concave vertex for faces and convex vertex for holes.

And thirdly, updating the intersection S must also be slightly modified. Since non-manifold vertices have two vertical edges, an endpoint of a segment in S may coincide with the beginning point of the next segment, although the two segments may belong to different faces. Therefore, it must be minded that the xor operation merges two segments only if they share an end and belong to the same face.

6. 3D boundary extraction

To extract the 3D boundary of an orthogonal polyhedron means to obtain the faces parallel to the three coordinate planes. Following an approach similar to [12], we may do three traversals in the three axis directions. Each traversal visits all planes P parallel to a coordinate plane that contain faces of the polyhedron (i.e. visits all *layers*), forms the lists of vertices (projected to 2D) of the polyhedron that lie on P, and calls the 2D algorithm that given this list extracts the faces on P. Before dealing with the different traversals of the polyhedron, it has to be decided which vertices are duplicated.

Let us show how the outlined approach works. Two-manifold orthogonal polyhedra have three kind of vertices, depending on the number of faces incident to them: V3, V4 and V6 (see Figure 5). V3 vertices are inserted in these lists once, whereas V6 vertices have to be inserted twice in each list, because if we analyse the faces incident to a V6 vertex parallel to one of the coordinate planes we see that topologically they correspond to a pseudomanifold 2D vertex, in any coordinate direction.

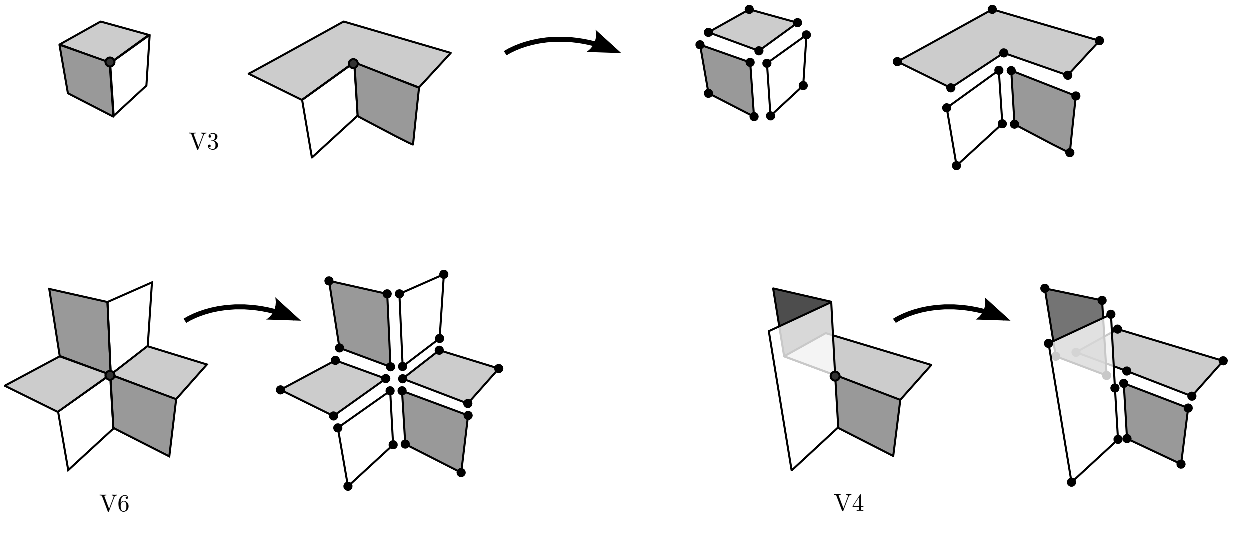


Figure 5: The 3 types of 3D orthogonal manifold vertices and the faces they give place to.

V4 vertices are the most particular case. The faces incident to a V4 vertex parallel to one of the coordinate directions also have a pseudomanifold configuration. But in the other two coordinate directions, there is only one face converging to the vertex, and these two faces contain two consecutive collinear edges (largest faces of the V4 vertex in Figure 5). However, notice that if the V4 vertices are always inserted twice in the lists, regardless of the coordinate axis we are traversing, the 2D algorithm forms polygons even if they contain collinear edges. Indeed, when the input list of vertices is sorted into Lxy and Lyx, two consecutive collinear edges will give place to a zero length edge in one of these lists, which are correctly managed by the sweep line algorithm since it is mainly based on topological tests, not geometric. [2] We only need, as a post-process, to traverse the polygons by removing duplicate vertices in a row, so that zero length edges are deleted.

Instead of performing these three traversals, a simpler approach which only needs three calls to the 2D extraction is possible. Note that if we form the lexicographically zxy-sorted list $Lzxy$ of all the vertices of the orthogonal polyhedron, a single call to the 2D function outputs the entire set of faces parallel to the Z plane. Being the first sorting criterion the z value, vertices with the same z are grouped in $Lzxy$. Then, the 2D sweep line will be performed successively with vertices on each plane parallel to z, and the alternate search will look for vertices in lists $Lzxy$ and $Lzyx$ as required. Needless to say, we have to adapt the 2D algorithm to handle 3D coordinates. In summary, all we have to do is to sort vertices using the six possible lexicographical orders into six lists $Lzxy, Lzyx, Lyxz, Lyzx, Lxyz, Lxzy$, and then use the 2D sweep line three times.

Notice that this approach may be slower than the original 3D idea that traverses all orthogonal plane layers, because while in one case the alternate search uses a single list with all the vertices, in the other a collection of smaller lists are used. However, the worst-case algorithmic complexity of both approaches is the same, as all vertices of a polyhedron may be located in only two planes.

There is one thing that remains unexplained concerning the 3D boundary extraction: output faces have to be oriented (i.e, polygon vertices have to be clockwise or counter-clockwise ordered) according to face normals. This is left to the next section, where we deal with the most general case, the 3D orthogonal pseudomanifolds.

7. Managing 3D pseudomanifolds

To extract the boundary of a 3D orthogonal pseudomanifold, we need that the algorithm handles 3D non-manifold vertices. The 2D reconstruction process is able to manage pseudomanifolds, i.e. 2D vertices such that, locally, two faces (or two portions of a face) converge to them. However, there are some 3D non-manifold configurations such that more than two faces meeting at a vertex lie on the same axis-aligned plane (up to four, see Figure 6). Therefore, the algorithm for 3D manifolds is not able to handle pseudomanifolds.

Nevertheless, notice that in any orthogonal non-manifold 3D vertex at most two faces meet sharing the same *oriented* plane. Therefore, we can compute a list of vertices for each oriented

[2]Strictly speaking, the only potentially conflictive geometrical test the 2D algorithm performs is the choice between two candidate edges in non-manifold 2D vertices. But since collinear 2D edges produced by V4 vertices do not meet at a non-manifold 2D vertex, this test is never applied to them.

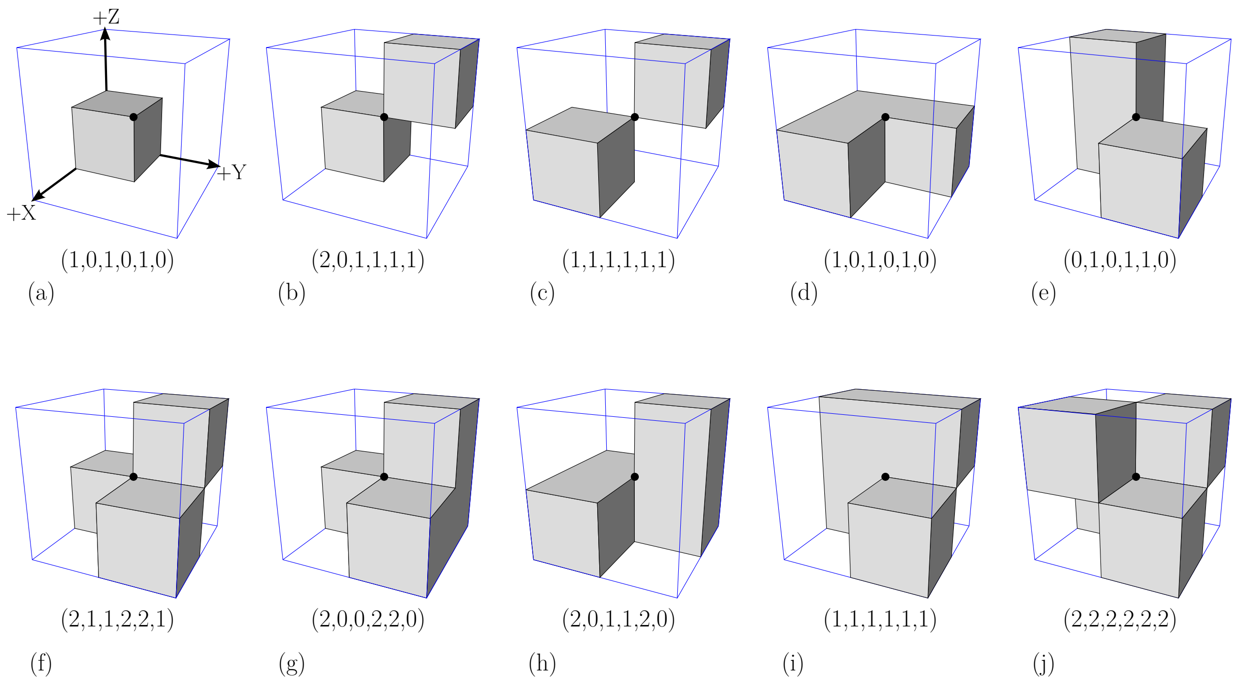


Figure 6: 3D vertices and the number of times they must be inserted in the 6 sorted lists. Printed values correspond to orientations $(+X, -X, +Y, -Y, +Z, -Z)$.

axis direction ($+X$, $-X$, $+Y$, $-Y$, $+Z$ and $-Z$) and proceed as in the manifold 3D case, applying 6 times the 2D boundary extraction algorithm. The main 3D algorithm needs as an input a list of vertices with six values associated to each vertex indicating the number of times (0, 1 or 2) the vertex must be repeated for each call (this is the additional information attached to every vertex in the general case). Except for the V4 vertices, these values coincide with the number of faces converging to the vertex that are oriented as each coordinate axis (“big” faces of V4 vertices count as two faces). Figure 6 shows manifold and non-manifold vertices and their corresponding 6 values. The 10 cases represented in the figure cover all possible vertex configurations; the 256 total cases can be deduced by either rotations and/or dual configurations of these 10 [22].

Since now a call to the 2D boundary extraction function is performed for each oriented axis, it is straightforward to orient the output set of polygons, by reversing either the holes or the external polygons depending on the axis orientation.

The worst-case time complexity of the 3D algorithm for pseudomanifolds is $\mathcal{O}(n \log n)$, being n the number of vertices of the pseudomanifold, because it requires six lexicographical sortings of the input list followed by six calls to the 2D boundary extraction algorithm. The space complexity for the 3D algorithms is $\mathcal{O}(n)$.

In order that the reader can check the validity of the proposed approach, we provide the source code that extracts the boundary of any 3D pseudomanifold. The program is published under a public license, see [26].

8. Conversion from other models

In this section, it has to be explained how the input format is obtained from different existing models. In particular, in order to apply the previous algorithm, the set of the vertices of a non manifold polyhedron and the classification to obtain their corresponding 6 values have to be computed from different enumeration models: the voxel model and the octree model; and from models based on vertex lists: the neighbourhood representation, the EVM model and the vertex list representation.

In case of a voxel model, the reconstruction of the B-Rep model needs a preprocess based on a traversal of all the voxel vertices. Each voxel vertex is analysed to know if it gives a vertex of the polyhedron and, in this case, the number of faces converging to the vertex. Moreover, as the voxels are usually stored in lexicographical order, the consistency between consecutive voxels can be used to compute the corresponding 6 values. Therefore, the time complexity depends linearly on the number of voxels. In most of the cases the number of polyhedron vertices is smaller than the number of voxels, thus the complexity of the construction of the input set gives the resulting complexity.

In case of an octree representation, the terminal nodes of the octree have to be visited, and the same analysis than in case of the voxel model is performed. Now, the complexity depends on the surface of the represented object, because that gives the number of nodes to analyse.

The neighbourhood representation [18] represents an orthogonal polyhedron by its set of vertices, and any vertex has attached its colour, that gives neighbouring information. In the 3D case, the colour of a vertex v gives information about the inclusion in the polyhedron of the 8 vertices of a box centered in v. Notice that the numbers of oriented faces incident in each vertex (to compute the corresponding 6 values) are straightforward obtained looking the colour of the neighbours and comparing them. The complexity of the input set construction is $\mathcal{O}(n)$, because it requires to traverse all the vertices, thus the B-Rep model is obtained in $\mathcal{O}(n \log n)$ time, where n is the number of vertices.

For the EVM [3], the boundary extraction method can be applied directly to 2D EVM because in 2D extreme and manifold vertices coincide. For 2D pseudomanifolds a preprocess that extracts non-manifold vertices is needed. 3D EVM can be converted into its complete set of oriented faces with pseudomanifold polygons, using EVM appropriate methods. Then, the 2D boundary extraction method can be applied to each oriented face. See Appendix B for a more detailed description of these methods.

The weighted vertex list model represents a d-dimensional scalar field as a set of weighted vertices [19]. Each vertex of the set v placed at $p(v)$ with weight $w(v)$ and coordinates p_i, $i = 1 \dots d$ modifies the scalar field by adding $w(v)$ to all points q such that $p(v) \leq q$, i.e. to all points q with coordinates $q_i, i = 1 \dots d$ such that $p_i \leq q_i, \forall i = 1 \dots d$. A set of vertices $V = \{v_1, \dots v_n\}$, where the position of vertex v_i is denoted by $p(v_i)$ and its weight by $w(v_i)$, assigns a scalar field Q_v to any point q, being $Q_v(q) = \sum_{p(v_i) \leq q} w(v_i)$. For convenience, the authors propose to store the set of vertices as a lexicographically sorted list. An orthogonal solid is represented by a vertex list that maps the interior of the solid to 1 and the outside to 0. Figure 1b shows an example of an orthogonal manifold represented as a weighted vertex list. Although it may seem that to compute the weight of a vertex we need to know all weights of the precedent

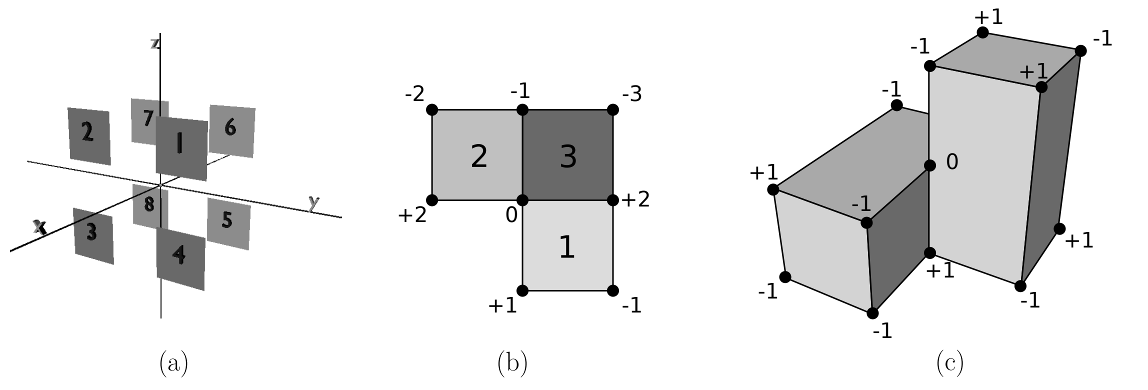


Figure 7: (a) Octant numbering. (b) and (c) two examples of models represented as a weighted vertex set that have a vertex with zero weight. (b) A 2D multivalued scalar field; (c) a 3D orthogonal polyhedron.

vertices, we remark that the weight can be calculated by examining the scalar field values in a neighbourhood of the vertex: assuming classical numbering of quadrants, in 2D the weight of any vertex is the sum of field values of the two even quadrants around the vertex minus the odd quadrant field values; similarly, in 3D, if octants are numbered as in Figure 7a, the weight of a vertex can be computed by adding the four even octant values and subtracting the odd octant values. In this way, for example, it is easy to deduce that non-manifold 2D vertices have weight +2 or -2.

Given that this model represents a polyhedron as a lexicographically sorted lists of weighted vertices, it seems very suitable to use our approach to transform from this model to a B-Rep. However, the authors state that this vertex representation is unique only if two restrictions are fulfilled: no two vertices may lie at the same point in the space, and no vertices may have zero weight [19]. Notice that in 3D some kind of vertices of a pseudomanifold have zero weight, in fact V4 vertices and one two kind of non-manifold vertices are zero weighted (cases (c), (h) and the dual configuration of (c) in Figure 6). Incidentally, in multivalued 2D objects, some points where there is a change in the value of the field also have zero weight (see Figure 7). This can easily be checked by summing and subtracting field values around the vertex. This implies that these zero weight vertices will not be in the vertex list. Therefore, we have a situation similar to the EVM model: our algorithm needs as input a sorted list with all vertices, but some kind of vertices are not explicit in the model. Recovering these missing zero weight vertices and also the 6 corresponding values of all vertices can be done in a very similar way than in the EVM case explained in Appendix B, using a traversal of vertex list that temporarily stores sections. Notice that the (zero weighted) missing vertices in this representation scheme are also missing in the EVM model.

9. Results

In order to check the complexity of our proposals, we have generated a number of random voxel models of increasing size, measuring the time spent by the part of the algorithm that ex-

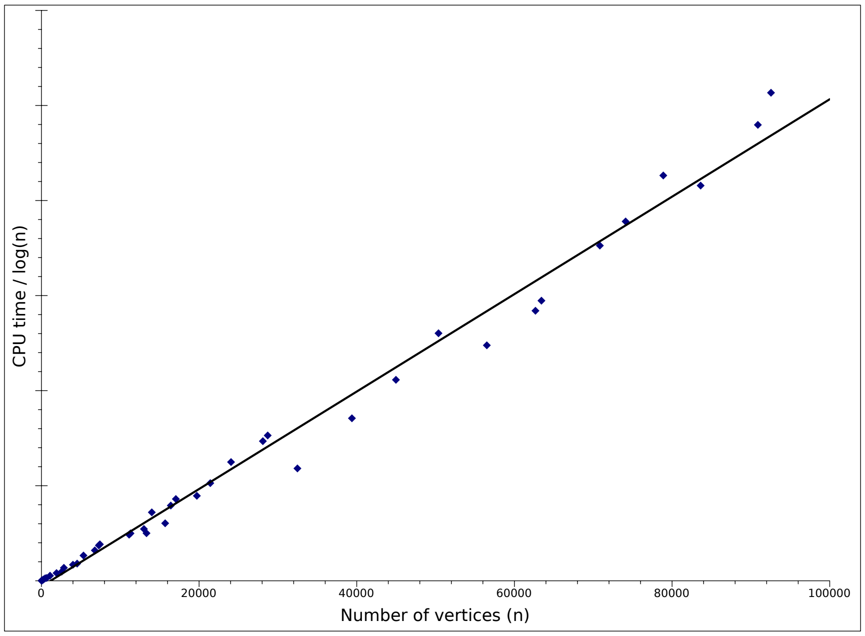


Figure 8: CPU time of our method for several 3D randomly generated pseudomanifolds.

tracts the B-Rep of the objects from the list of vertices. Figure 8 shows this CPU time divided by $\log(n)$ depending on the number of vertices of the model, n. A linear interpolation of the data is also plotted. Besides to checking that, in the average case, our algorithm is linearithmic, these tests have also been useful to ensure the robustness of the proposal, since large random models include a large number of different non-manifold vertex cases and hole-face configurations.

Figure 9 shows a small voxel model and the faces formed by our proposal on the the six possible axis orientations. The object in the figure has 42 voxels, and its B-rep is composed of 112 faces and 140 vertices. Only two faces have a hole.

We also converted several real voxel models to B-Rep. Figure 10 shows a rendering of four of these models. In these cases the CPU time of the conversion process is ruled by the first step, that

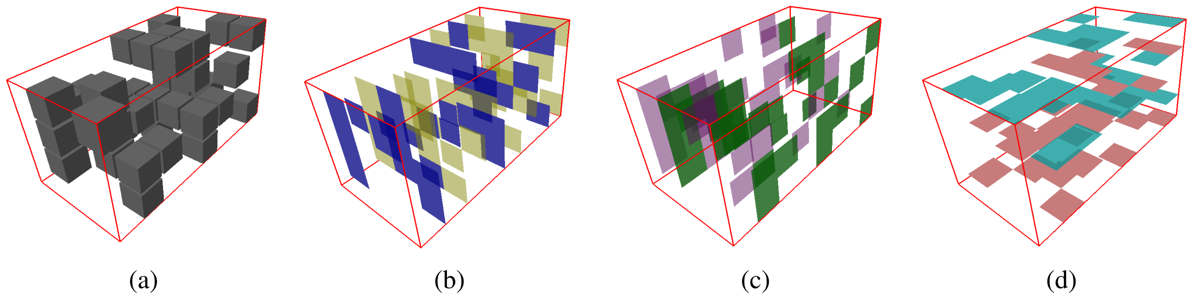
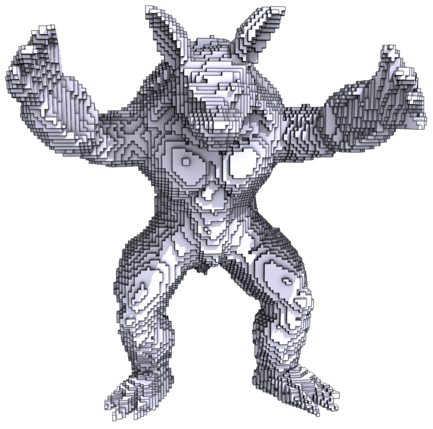
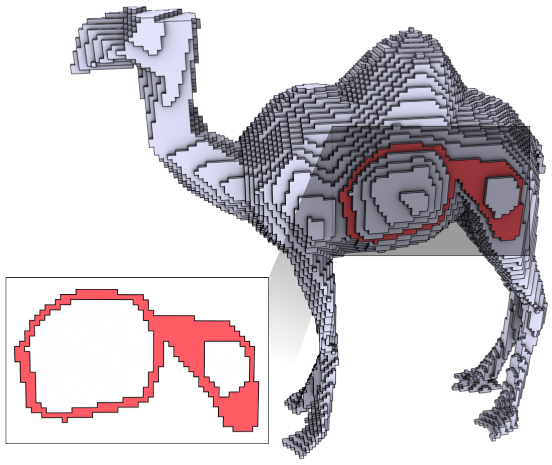


Figure 9: (a) A voxels model and (b,c,d) its boundary faces in the three axis coordinate directions. Faces are drawn semi-transparent and using different colours depending on their orientation.

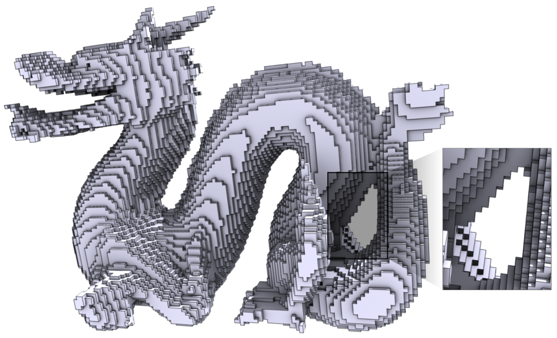
is to say, the voxel traversal to obtain the list of vertices. Typically, in real models the number of voxels is hundreds of times higher than the number of vertices, and only about 5% of the vertices of the object are non-manifold. In the figure, details are zoomed to show that the algorithm is able to correctly manage any pseudomanifold, including faces with holes, non-manifold vertices configurations, solids with several connected components and objects of any genus.



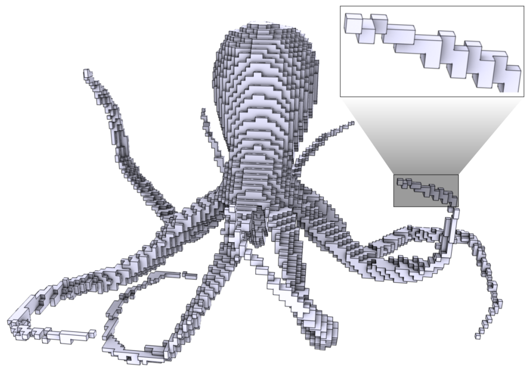
(a)



(b)



(c)



(d)

Figure 10: Rendering of four B-Rep orthogonal models output by the algorithm.

10. Conclusions

In this paper, we studied the problem of reconstructing an orthogonal pseudomanifold polygon or polyhedron given its set of vertices and, in 3D, the information corresponding with the number of faces that meet at the vertex. The B-Rep representation of any orthogonal pseudomanifold polygon and polyhedron can be obtained in $\mathcal{O}(n \log n)$ time, where n is the number of vertices. Orthogonal polygons and polyhedra are intensively studied in Computer Graphics, Solid Modeling and Computational Geometry, thus the presented algorithm can be applied to obtain a valid representation for different existing approaches which needs a B-Rep model. Although our approach can be used to convert spatial enumeration models as well as vertex lists based models, it is best suited for the second category of models, because in the spatial enumeration models the number of elements rules the global complexity of the whole conversion process. However, we also recommend it for the first category, since our approach correctly handles all kind of non-manifold vertices and outputs a complete hierarchical B-Rep model that includes all hole-face relationships.

11. Acknowledgements

This work has been partially supported by the project TIN2008-02903 of the Spanish government and by the IBEC (Bioengineering Institute of Catalonia).

References

[1] H. E. Cline, W. E. Lorensen, R. J. Herfkens, G. A. Johnson, G. H. Glover, Vascular morphology by three dimensional magnetic resonance, Magnetic Resonance Imaging 7 (1989) 45–54.

[2] A. Rosenfeld, T. Kong, A. Wu, Digital surfaces, CVGIP: Graphical Models and Image Processing 53 (1991) 305–312.

[3] A. Aguilera, D. Ayala, Converting orthogonal polyhedra from extreme vertices model to B-Rep and to alternating sum of volumes, Computing Supplement 14 (2001) 1–28.

[4] G. J. Grevera, J. K. Udupa, D. Odhner, An order of magnitude faster isosurface rendering in software on a PC than using dedicated, general purpose rendering hardware, IEEE Transactions on Visualization and Computer Graphics 6 (2000) 335–345.

[5] Y. Livnat, X. Tricoche, Interactive point-based isosurface extraction, in: Proc. IEEE Visualization, 2004, pp. 457 – 464.

[6] C. A. Dietrich, C. E. Scheidegger, J. Schreiner, J. Comba, L. Nedel, C. T. Silva, Edge transformations for improving mesh quality of marching cubes, IEEE Transactions on Visualization and Computer Graphics 15 (1) (2009) 150 – 159.

[7] A. Greβ, R. Klein, Efficient representation and extraction of 2-manifold isosurfaces using kd-trees, Graphical Models 66 (2004) 370 – 397.

[8] S. Schaefer, T. Ju, J. Warren, Manifold dual contouring, IEEE Transactions on Visualization and Computer Graphics 13 (3) (2007) 610 – 619.

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

[10] J. O'Rourke, Unfolding orthogonal polyhedra, in: Surveys on discrete and computational geometry: twenty years later: AMS-IMS-SIAM Joint Summer Research Conference, Vol. 453, Amer Mathematical Society, 2008, p. 307.

[11] J. Martinez, M. Vigo, N. Pla-Garcia, Skeleton computation of orthogonal polyhedra, Computer Graphics Forum 30 (5) (2011) 1573–1582.

[12] C. Montani, R. Scopigno, Quadtree/octree to boundary conversion, in: Graphics Gems II, 1991, pp. 202–218.

[13] K. Karunakaran, R. Shringi, Octree-to-BRep conversion for volumetric NC simulation, The International Journal of Advanced Manufacturing Technology 32 (2007) 116–131.

[14] H. J. A. M. Heijmans, Connected morphological operators for binary images, Computer Vision and Image Understanding 73 (1999) 99–120.

[15] S. C. Park, B. K. Choi, Boundary extraction algorithm for cutting area detection, Computer-Aided Design 33 (2001) 571–579.

[16] G. Damiand, Topological model for 3d image representation: Definitionand incremental extraction algorithm, Computer Vision and Image Understanding 109 (2008) 260 – 289.

[17] I. Gargantini, G. Schrack, A. Kwok, Reconstructing multishell solids from voxel-based contours, Computer-Aided Design 26 (1994) 293–301.

[18] O. Bournez, O. Maler, A. Pnueli, Orthogonal polyhedra: Representation and computation, in: Hybrid Systems: Computation and Control, LNCS 1569, 1999, pp. 46–60.

[19] C. Esperança, H. Samet, Vertex representations and their applications in computer graphics, The Visual Computer 14 (5/6) (1998) 240–256. doi:10.1007/s003710050138.

[20] J. O'Rourke, Uniqueness of orthogonal connect-the-dots, in: Computational Morphology, North-Holland, 1988, pp. 97–104.

[21] C. Esperança, H. Samet, Representing orthogonal multidimensional objects by vertex lists, in: Aspects of Visual Form Processing: Proceedings of the 2nd International Workshop on Visual Form (IWVF2), World Scientific, Capri, Italy, 1994, pp. 209–220.

[22] A. Aguilera, Orthogonal polyhedra: Study and application, Ph.D. thesis, LSI-Universitat Politècnica de Catalunya (1998).

[23] T. Biedl, B. Genç, Reconstructing orthogonal polyhedra from putative vertex sets, Computational Geometry 44 (2011) 409–417.

[24] J. Rossignac, D. Cardoze, Matchmaker: manifold BReps for non-manifold r-sets, in: Proceedings of the fifth ACM symposium on Solid modeling and applications, 1999, pp. 31–41.

[25] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, Computational Geometry: Algorithms and Applications, Springer, 2008.

[26] M. Vigo, Orto-brep (Nov. 2011).
URL `http://devel.cpl.upc.edu/orto-brep/`

[27] Python Software Foundation, Python documentation (1990-2011).

[28] P. H. Chou, Algorithm Education in Python, in: Proceedings of Python 10, 2002.

[29] H. Fangohr, A Comparison of C, MATLAB, and Python as Teaching Languages in Engineering, in: Computational Science - ICCS 2004, Springer Berlin / Heidelberg, 2004.

A. A Python implementation

A Python implementation of the 2D algorithm is next presented. We choosed Python [27] because it is a high level language close to natural language, resembling pseudo code. Many authors noticed the advantages of scripting languages (see [28] and [29], among others).

```python
def boundary_extract_2d(lv):
    lxy = sorted(lv)
    lyx = sorted(lv, key = lambda v : (v[1], v[0]) )
    lpols = []          # polygons formed
    holes = {}          # holes of each face
    s = []              # the intersection
    face_of_edge = {}   # edge->face relationship (marks)
    for i in range(0, len(lxy), 2):
        (vi, vf) = (lxy[i], lxy[i+1])
        if (vi, vf) not in face_of_edge:     # if edge is not marked
            idpol = len(lpols)
            idface = update_intersection(s, vi, vf, idpol)
            is_hole = idface >= 0
            pol = form_polygon(lxy, lyx, i, is_hole)
            lpols.append( (pol, is_hole) )
            if is_hole:
                holes[idface].append(idpol)
            else:
                idface = idpol
                holes[idface] = []
            set_face_edges(pol, idface, face_of_edge)
        else:
            idface = face_of_edge[vi, vf]
            update_intersection(s, vi, vf, idface)
    return lpols, holes
```

In the Python code we assume that function boundary_extract_2d receives as parameter a list of vertices and returns two values: a list and a Python dictionary. The elements of the list returned are pairs (polygon, flag), being the polygon a list of vertices and the flag a boolean to indicate if the polygon is either external or a hole. The dictionary stores the list of holes for each face.

The call to update_intersection, as its name suggests, updates the s with a new vertical segment $e = (v_i, v_f)$, merging the segments in s with e. Since the segments in s must be labelled with to their corresponding face, the identifier of the face must be supplied as an argument to the function. In the case that the update finds that e falls completely inside one of the segments of s, segment e is identified as part of a hole, and the identifier of the face that e belongs to is returned; otherwise, form_polygon returns a negative value.

Function form_polygon, given the lists lxy and lyx, constructs a polygon starting from i-th segment of lxy. The polygon is formed using the alternate search of end vertices of the edges in the lists as described previously.

We use a Python dictionary (face_of_edge) to maintain the edge-face membership, the marks mentioned. Function set_face_edges simply traverses a polygon pol just formed storing this information for all vertical edges of pol.

Notice that as designed the function boundary_extract_2d computes all polygons clockwise oriented, regardless if they are holes or not. But thanks to the booleans included in the output, holes can easily be identified and counter-clockwise oriented by simply reversing the list of its vertices.

Remark that even though it would have resulted in an somewhat simpler algorithm, we do not separate the polygon reconstruction from the face-hole classification processes. Later, when we deal with pseudomanifolds this will not be possible: the reconstruction will be done differently depending on whether the polygon is external or a hole (that is the reason why function form_polygon receives as last parameter a flag indicating if a hole must be formed, see Section 5). Also remark that the main iteration could be finished when all polygons are formed – that is to say, when all vertical edges are marked – instead of traversing all edges in Lxy. In any case, none of these two changes affect the worst-case complexity of the algorithm.

The Python code is not optimised in terms of time complexity, because it uses a list to store the intersection S, a dictionary to store marks, etc. However, the code faithfully follows the proposed scheme and can be easily adapted to work with the appropriate data structures that optimise efficiency.

B. Conversion from EVM

EVM is a model suitable for orthogonal manifolds and pseudomanifolds. It stores in a lexicographically sorted list only a subset of vertices (extreme vertices) which are the ending vertices of maximal uninterrupted segments (see Figure 11). In 2D there are two possible sortings, Lxy and Lyx and in 3D there are the corresponding six sortings.

Let P be an EVM. A *cut*, $C(P)$, is the set of vertices of P lying on a plane (line) perpendicular to a main axis. A *slice* is the region between two consecutive cuts which is represented by its *section*, $S(P)$. See Figure 11. Sections can be computed from cuts and vice-versa:

$$\begin{array}{l} \overline{S_i(P)} = \overline{S_{i-1}(P)} \otimes \overline{C_i(P)}, i = 1 \ldots n, S_0(P) = S_n(P) = \emptyset \\ \overline{C_i(P)} = \overline{S_{i-1}(P)} \otimes \overline{S_i(P)}, i = 1 \ldots n \end{array} \tag{1}$$

where $\overline{C_i(P)}$ and $\overline{S_i(P)}$ denote the projections of $C_i(P)$ and $S_i(P)$ onto a main plane (line) parallel to them, n is the number of cuts and $\otimes$ denotes the regularised xor operation.

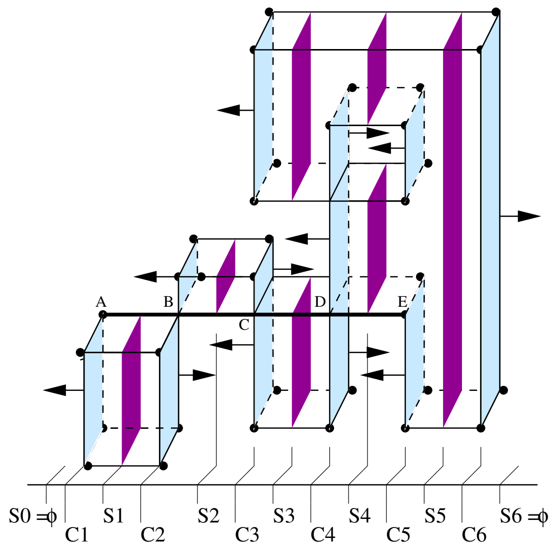


Figure 11: An orthogonal pseudomanifold. A and E are extreme vertices and B, C and D are non-extreme. Cuts and sections for one direction are shown.

Applying the definition of xor, the last equation can be expressed as:

$$\overline{C_i(P)} = (\overline{S_{i-1}(P)} -^* \overline{S_i(P)}) \cup (\overline{S_i(P)} -^* \overline{S_{i-1}(P)}) = \overline{FD_i(P)} \cup \overline{BD_i(P)} \qquad (2)$$

thus, any cut, $C(P)$, is decomposed into two terms named *forward difference* ($\overline{FD_i(P)}$) and *backward difference* ($\overline{BD_i(P)}$). $FD_i(P)$ is the set of faces (edges) of $C_i(P)$ whose normal vector points to one side of the main axis perpendicular to $C_i(P)$, while $BD_i(P)$ is the set of faces (edges) pointing to the opposite side.

It exists a conversion method from EVM to B-Rep [3]. This method is based on Expression 2 that is applied in 3D to obtain oriented faces and then to each 2D forward and backward differences to obtain oriented edges. As said in the introduction, then a brute force method of quadratic complexity is applied to obtain the hole-face inclusion relationships using point-contour inclusion tests. However, this method presents a flaw when dealing with V4 vertices. These vertices present a 2D pseudomanifold configuration in one main direction, say X for instance, and the application of Expression 2 separates both faces, and shows up the V4 vertex, because they have opposite direction. But in the other two main directions (biggest faces of the V4 vertex in Figure 5) this vertex does not appear. Then, in the resulting B-Rep model, these vertices only appear in the contour of two faces instead of four.

Although the method presented in this paper has some similarities with this EVM to B-Rep conversion method, it has also relevant differences and improves it in several aspects. Both methods perform a traversal in each main direction and a 2D sweep along the planes afterwards. However, in the presented method, this sweep obtains the complete contours with the hole-face inclusion relationships while in the other one only the edges were obtained. These methods use different models and ways to obtain the orientation of faces. In the presented method the input model is a list of all vertices with 6 values at each vertex that give orientation information while

EVM is a list of a subset of vertices and uses Expression 2 to compute the orientation. Finally, the EVM to B-Rep conversion method presents the V4 vertices related flaw that the presented method solves.

Now we explain how to obtain the input data needed to apply the presented boundary extraction methods from an EVM.

In a 2D orthogonal manifold all vertices are extreme and, therefore, the boundary extraction method can be applied directly.

For 2D pseudomanifolds non-extreme vertices coincide with non-manifold ones. As they are not in the EVM, they must be detected. We will use the following property that can be proved by exhaustion. Given Lxy, any vertex of a section such that its y coordinate is included in the open interval defined by the y coordinates of any vertical edge of the next cut is a non-manifold vertex. Therefore, a preprocess is performed that computes sections sequentially with expression (see Equation 1) and detects non-manifold vertices, which are inserted twice. This preprocess has the same complexity that the boundary extraction method because it actually performs the same operations. Note that a cut is the set of vertical edges belonging to the same vertical line and that the status data structure S coincides with a section when the last edge of a vertical line has been processed.

For 3D polyhedra, the method that converts from EVM to B-Rep [3] is applied to obtain FD and BD for each main direction and then we can apply to each of these six sets the 2D preprocess explained in the previous paragraph and the 2D boundary extraction method presented in Section 4. However, the application of this method also presents the flaw concerning to V4 vertices that have to be have to be detected and inserted appropriately beforehand.

Let X^+ be the set of vertices belonging to those faces with normal vector $(1, 0, 0)$ and let X^-, Y^+, Y^-, Z^+ and Z^- be defined analogously. Those V4 vertices that appear, in the set $X = X^+ \cup X^-$, will not be in the sets $Y = Y^+ \cup Y^-$ and $Z = Z^+ \cup Z^-$. Therefore, we can detect them by performing an xor operation between sets X and Y (or Z). Then we have to insert these vertices in the sets that contain the other faces converging to the V4 vertex, i.e., either in Y^+ or Y^- and either in Z^+ or Z^-. In the two sets that contain these faces, the V4 vertex must go in an even position because it splits one edge, while in the other two sets it would go in an odd position because it doesn't split any edge. Therefore, the insertion of V4 vertices is performed based on these facts. V4 vertices that appear in sets Y and Z are processed in the same way. Note that this process also would solve the flaw of the EVM to B-Rep mentioned conversion method.