On the validation of solids represented with the international standards for geographic information

Hugo Ledoux
h.ledoux@tudelft.nl

Hugo Ledoux (2014). On the validation of solids represented with the international standards for geographic information. *Computer-Aided Civil and Infrastructure Engineering*, 28(9):693–706.
DOI: http://dx.doi.org/10.1111/mice.12043

The code of the project is available at
https://github.com/tudelft3d/val3dity

The international standards for geographic information provide unambiguous definitions of geometric primitives, with the aim of fostering exchange and interoperability in the geographical information system (GIS) community. In two dimensions, the standards are well-accepted and there are algorithms (and implementations of these) to *validate* primitives, i.e. given a polygon, they ensure that it respects the standardised definition (and if it does not a reason is given to the user). However, while there exists an equivalent definition in three dimensions (for solids), it is ignored by most researchers and by software vendors. Several different definitions are indeed used, and none is compliant with the standards: e.g. solids are often defined as 2-manifold objects only, while in fact they can be non-manifold objects. Exchanging and converting datasets from one format/platform to another is thus highly problematic. I present in this paper a methodology to validate solids according to the international standards. It is hierarchical and permits us to validate the primitives of all dimensionalities. To understand and study the topological relationships between the different parts of a solid (the shells) the concept of Nef polyhedron is used. The methodology has been implemented in a prototype, and I report on the main engineering decisions that were made and on its use for the validation of real-world three-dimensional datasets.

1 Introduction

To facilitate and encourage the exchange and interoperability of geographical information, the ISO[1] and the OGC[2] have developed in recent years standards that define what the basic geographical primitives are (the abstract specifications [ISO, 2003]), and also how they can be represented

[1] International Organization for Standardization: www.iso.org
[2] Open Geospatial Consortium: www.opengeospatial.org

in a computer (the implementation specifications [OGC, 2007, 2006]). While the abstract definitions for the primitives are not restricted to two dimensions (2D), most of the efforts for the representation and storage of the geographical primitives have been done only in 2D. However, in recent years, with the adoption of CityGML as an international standard [OGC, 2012], the amount of available datasets with three-dimensional (3D) primitives is increasing rapidly. Although the topic might appear trivial—"a polygon is simply a polygon, no?"—it is in practice a problem and a topic of research. As van Oosterom et al. [2004] discuss: (i) there are several differences between the standards and their implementations in different systems, and (ii) in extreme cases, related to the floating-point arithmetic used by computers, parts of polygons can "collapse" and become invalid.

Having unambiguous definitions for the geometric primitives is important to foster interoperability, but perhaps as important is to have algorithms and tools to *validate* primitives. That is, given a specific primitive, we need to check if it respects the standardised definitions. Most GIS operations (e.g. calculation of the area of polygons; creation of buffers; conversion to other formats; Boolean operations such as intersection, union, etc.) indeed require that the input primitives be valid, otherwise the output of the operation is not guaranteed.

The validation rules for 2D primitives are well-defined and there are several (open-source) implementations of these: JTS[3] and GEOS[4] are the most well-known examples, and are used by different software packages. However, for primitives in 3D (primitives representing a volume, called either a solid or a polyhedron), there are no such validation rules, implementation specifications or known implementations. Indeed, as described in Section 3, the algorithms and tools currently available all use different definitions, and these are often simpler and more restrictive than these of the international standards. The two principal restrictions are: (i) holes in faces and voids inside solids are not considered; (ii) solids are often defined as 2-manifold objects. However, holes, cavities and non-manifold are allowed according to the ISO and the OGC international standards, and furthermore, they are necessary to represent all the real-world objects.

I present in this paper a methodology to validate solids against the definition of the ISO/OGC. I first give in Section 2 the definition of an ISO/OGC solid (which forms the basis for the implementation specifications; these are presently lacking), and then explain in Section 3 why current methods in GIS and computer-aided design (CAD) are not appropriate. As described in Section 4, the methodology I propose is hierarchical (the primitives of different dimensionalities are validated separately, which ensures that the user does not get cascading errors, i.e. errors caused by errors in lower dimensionality primitives), and it requires the use of both 2-manifold and non-manifold data structures and algorithms. Nef polyhedra, which are defined in Section 4.3, are used to inspect the topological relationships between the different boundaries that a solid can have, certain topological configurations are allowed by the standards while others are not (the allowed configurations are described in Section 2). I also present in Section 5 one implementation of the proposed methodology, and describe in Section 6 some experiments that were run with unit test solids (a set of solids containing extreme cases) and real-world examples of 3D city models.

2 What is a solid? And the implications for its validation

There is no single definition for a solid or a polyhedron (notice that these two terms are often used interchangeably in the scientific literature). This is best illustrated by Grünbaum [2003] who states that even in the field of mathematics opinions differ as to what constitutes the term 'polyhedron'. Some use it only for a regular polyhedron, or only for a convex one, and some consider non-planar faces as part of the definition. De Berg et al. [2000] characterise the term as "difficult to define", and give a simple definition that is probably the most common one: a polyhedron is a 3D solid bounded by planar faces. The bounding faces are thus planar surfaces embedded in $\mathbb{R}^3$, the three-dimensional Euclidean space, and the bounding surfaces form a *closed two-dimensional manifold* (or 2-manifold for short). A 2-manifold is a topological space that is topologically equivalent to $\mathbb{R}^2$. An obvious example is the surface of the Earth, for which near to every point the surrounding area is topologically equivalent to a plane. Representing and storing a 2-manifold, even in $\mathbb{R}^3$, can be done with data structures that are intrinsically 2D since

[3]Topology Suite: `www.vividsolutions.com/jts`

[4]Geometry Engine, Open Source: `trac.osgeo.org/geos`

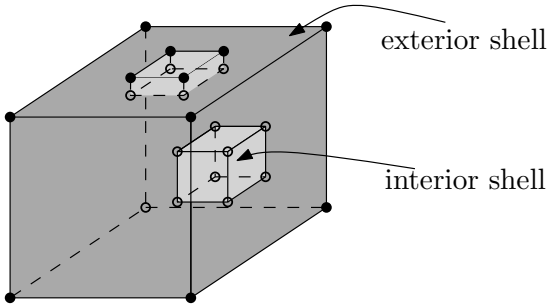


Figure 1: One solid which respects the international definition. It has one exterior shell and one interior shell (a cavity).

each edge is guaranteed to have a maximum of two incident faces [Kettner, 1999]. Examples of popular 2D data structures are the half-edge [Mäntylä, 1988], the quad-edge [Guibas and Stolfi, 1985], and the doubly-connected edge list (DCEL) [Muller and Preparata, 1978]; all of these store the edge of a polyhedron as the atom, with links to its adjacent edges and incident faces.

However, the ISO's definition of a solid is broader than that of a 2-manifold, and to permit us to represent all the real-world features. The following three definitions, taken from ISO [2003], summarise the main differences[5].

Definition 1 *A GM_Solid is the basis for 3-dimensional geometry. The extent of a solid is defined by the boundary surfaces. The boundaries of GM_Solids shall be represented as GM_SolidBoundary.*

Definition 2 *A GM_Shell is used to represent a single connected component of a GM_SolidBoundary. It consists of a number of references to GM_OrientableSurfaces connected in a topological cycle (an object whose boundary is empty). [...] Like GM_Rings, GM_Shells are simple.*

Definition 3 *A GM_Object is* simple *if it has no interior point of self-intersection or self-tangency. In mathematical formalisms, this means that every point in the interior of the object must have a metric neighbourhood whose intersection with the object is isomorphic to an n-sphere, where n is the dimension of this GM_Object.*

Since shells are *simple*, they are in fact 2-manifold objects.

Figure 1 shows a solid that respects that definition. First observe that the solid is composed of two shells (its boundaries), one being the exterior and one being the interior shell. The exterior shell has eleven planar surfaces, and the interior one six. An interior shell creates a cavity in the solid—cavities are also referred to as "voids" or holes in a solid. A solid can have no inner shells, or several. Observe that a cavity is not the same as the hole in a torus—a torus is represented with one exterior shell having a genus of 1 and has no interior shell. The boundary of each shell of a solid is a representation of a 2-manifold, but it should be noticed here that since shells are formed of planar surfaces, the ISO definition of a surface is used and this states that a polygon can have inner boundaries. The boundaries of a polygon are defined as rings, thus a hole in a face is referred to as an interior ring. Observe that the top face of the solid in Figure 1 has one inner ring, but that other surfaces "fill" that hole so that the exterior shell is "watertight". Several disciplines ignore holes because they are not necessarily needed, and because they complicate the representation of a 2-manifold: if a graph-based data structure is used, then the graph becomes unconnected.

From a point-set topology point-of-view, a solid is a set of points $S \subseteq \mathbb{R}^3$. A boundary of S, denoted ∂S, is a shell. It should be observed here that while a shell is used to represent a surface embedded in $\mathbb{R}^3$, when referring to a shell from a point-set topology point-of-view, we refer to the *volume* that the boundary contains. That is, let $H \subseteq \mathbb{R}^3$ be a shell, ∂H refers to the boundary of the shell, and H^o to its interior. H' refers to the complement of the shell (its exterior).

According to the ISO abstract specifications, the different boundaries of a solid are allowed to interact with each other, but only under certain circumstances. To understand these, we have to generalise to 3D the implementation specifications defined in 2D by the OGC (since they do not exist yet in 3D). Figure 2 shows the six assertions that have to be true for a 2D polygon to be

[5] All the geometric objects have the prefix 'GM_'

1. Polygons are topologically closed;
2. The boundary of a Polygon consists of a set of LinearRings that make up its exterior and interior boundaries;
3. No two Rings in the boundary cross and the Rings in the boundary of a Polygon may intersect at a Point but only as a tangent, e.g.

$$\forall P \in Polygon, \forall c1, c2 \in P.Boundary(), c1 \neq c2,$$

$$\forall p, q \in Point, p, q \in c1, p \neq q, [p \in c2 \Rightarrow q \notin c2];$$

4. A Polygon may not have cut lines, spikes or punctures e.g. :

$$\forall P \in Polygon, P = P.Interior.Closure;$$

5. The interior of every Polygon is a connected point set;
6. The exterior of a Polygon with 1 or more holes is not connected. Each hole defines a connected component of the exterior.

Figure 2: The six assertions for the validity of a polygon [OGC, 2006, pages 27–28].

valid. Observe that all of them, except the third one, generalise directly to 3D since a point-set topology nomenclature is used. The only modifications needed are that, in 3D, polygons become solids, rings become shells, and holes become cavities.

To further explain what the assertions are in 3D, Figure 3 shows 12 solids, some of them valid, some not; all the statements below refer to solids in this figure. The first assertion of the OGC means that a solid must be closed, or 'watertight' (even if it contains interior shells). The solid s_1 is thus not valid but s_2 is since the hole in the top surface is 'filled' with other faces.

The second assertion implies that each shell must be simple, i.e. that it is a 2-manifold.

The third assertion means that the boundaries of shells can intersect each others, but the intersection between the shells can only contain primitives of dimensionality 0 (vertices) and 1 (edges). If a surface or a volume is contained, then the solid is not valid. The solid s_3 is an example of a valid solid: it has two interior shells whose boundaries intersect at one point (at the apexes of the tetrahedra), and the apex of one of the tetrahedra is coplanar with the exterior shell. If the interior of the two interior shells intersects (as in s_4) the solid is not valid; this is also related to the sixth assertion stating that each cavity must define one connected component: if the interior of two cavities are intersecting they define the same connected component. Notice also that s_5 is not valid since one surface of its cavity intersects with one surface of the exterior shell (they "share a surface"); s_5 should be represented with one single exterior shell (having a 'dent'), and no interior shell.

The fourth assertion states that a shell is a 2-manifold and that no dangling pieces can exist (such as that of s_6); it is equivalent to the *regularisation* of a point-set in $\mathbb{R}^3$.

The fifth assertion states that the interior of a solid must form a connected point-set (in $\mathbb{R}^3$). Consider the solid s_7, it is valid since its interior is connected and it fulfils the other assertions; notice that it is a 2-manifold but that unlike other solids in Figure 3 (except s_8) its genus is 1. If we move the location of the triangular prism so that it touches the boundary of the exterior shell (as in s_8), then the solid becomes invalid since its interior is not connected anymore, and also since its exterior shell is not simple anymore (2 edges have 4 incident planar faces, which is not 2-manifold). It is also possible that the interior shell of a solid separates the solid into two parts: the interior shell of s_9 is a pyramid having four of its edges intersecting with the exterior shell, but no two surfaces are shared, thus these interactions are allowed. However, the presence of the pyramid separates the interior of the solid into two unconnected volumes (violating assertion 5); for both s_8 and s_9, the only possible valid representation is with two different solids.

Notice also that for a solid to be valid, all its lower-dimensionality primitives must be valid. That is, each surface of the shells (represented with polygons) has to be individually valid according to the assertions in Figure 2. Since these are embedded in $\mathbb{R}^3$, they first have to be projected to a plane. An example of an invalid surface would be one having a hole (an inner ring) overlapping the exterior ring (see s_{10}).

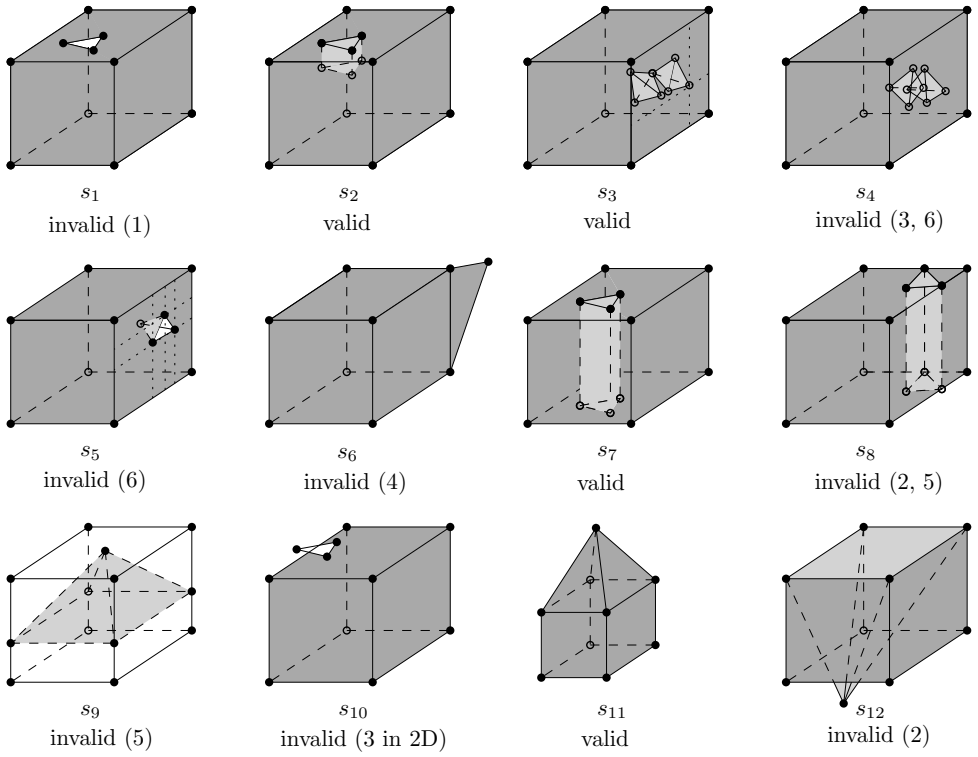


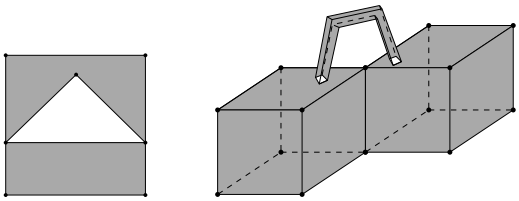
Figure 3: Twelve solids with the numbers in parentheses next to each indicating which OGC assertions are broken if invalid. For solid s_9, which is conceptually similar to the 2D polygon in Figure 4a, the colour of the exterior shell is not shown to highlight the interior shell.

The implications of the definition for the validation. To implement the validation rules and assertions in 3D, appropriate data structures must be used. Structures for 2-manifold modelling are not appropriate since non-manifold objects must be represented and checked. Furthermore, the data structure must permit us to represent and detect *volumes* so that the fifth assertion in Figure 2 can be tested; 2D structures such as the half-edge are not appropriate. The methodology described in Section 4 uses one such data structure.

It should also be noticed that when validating a solid both the combinatorial consistency and the geometric consistency of the representation should be valid. A solid such as s_{11} is valid, but if the location of only one of its vertices is modified (for instance if the apex of the pyramid of s_{11} is moved downwards to form s_{12}) then it becomes invalid. Both s_{11} and s_{12} can be represented with a graph having exactly the same topology (which is valid for both), but if we consider the geometry then the latter solid is not valid since its exterior shell is not simple. Enforcing simplicity requires calculating the intersections between the surfaces.

3 Related work

Many application domains where solids are modelled use their own definition of a solid, and often this definition is driven by the data structure used. In the computer-aided design (CAD) world, when a boundary representation (*b-rep*) is used, often 2-manifoldness is enforced. An example is Mäntylä [1988] who uses half-edges and Euler operators to construct and enforce the validity of a solid. The validity constraints are such that if a non-manifold solid needs to be represented, the geometry of the solid is slightly modified so that it becomes a 2-manifold (the triangular prism of the solid s_8 in Figure 3 would not touch the exterior shell along one of the 2 edges for instance, but be infinitesimally close to it). Fortune [1997] takes this approach one step further and performs *symbolic perturbations* of the equation of the planes bounding a non-manifold polyhedron so that the representation is 2-manifold. The modified polyhedron contains 0-volume parts, which need to be processed with care depending on the application.



(a) (b)

Figure 4: **(a)** A polygon in 2D whose inner ring separates the polygon (and it is thus not valid), the polygon is conceptually the same as the solid s_9 in Figure 3. **(b)** A so-called handle is added to connect the interior of two cubes sharing an edge.

In engineering applications where laser-scanners are used, often triangles are used as the only primitive for surfaces, and no interior shells are allowed; this is primarily because the size of the models can be large and one wants to ensure efficiency. To ensure validity of such models, and to repair them, different techniques exist (see for instance Nooruddin and Turk [2003] or Liepa [2003]).

In GIS-related applications, the definitions are also more restrictive than these of the international standards. Bogdahn and Coors [2010] and Wagner et al. [2012] discuss the validation of solids for city modelling, but do not consider holes in surfaces and totally omit that interior shells are possible. Gröger and Plümer [2011] give axioms to validate 3D city models, but also do not consider holes in primitives of dimensions 2 and 3; what they define as solids are in fact shells without holes in surfaces. I use their work on the validity of shells as a building block of the methodology presented in this paper.

Most commercial GIS companies also ignore interior shells, ESRI (with ArcGIS 10) and Bentley being two examples. Oracle Spatial considers interior shells in their validation function, but do not allow holes in surfaces [Oracle, 2012]. Also, while they claim to validate according to the international rules, in practice there are several inconsistencies since they use a graph-approach and perform graph-traversal algorithms to validate, see Kazar et al. [2008]. This approach is suitable for 2-manifold objects, but for solids having interior shells interacting with other shells it is not sufficient. One consequence is that it is impossible to detect that the solid s_9 in Figure 3 is not valid (since the configuration of the interior shell makes it such that the solid should be split in two solids). Observe that the configuration is akin to that in Figure 4a, but in 3D. In 2D, a planar graph can represent the topological relationships between the rings and permits us to detect that the polygon is not connected, but the same data structure cannot be used in 3D. We need one that permits us to represent non-manifold cases, and that permits us to navigate and identify connected parts.

To overcome the problems with a graph-based approach, a space-filling data structure should be used. Verbree and Si [2008] decompose the solid into tetrahedra and use the properties of the tetrahedralization to perform validation. However they do not consider holes in faces, interior shells and do not discuss how to verify that a shell is a 2-manifold. Ledoux et al. [2009] extend that methodology to solids as defined in Section 2, but their work has not been implemented.

Finally, Thompson and van Oosterom [2011] give axioms for the validation of solids (taking into account holes in faces and interior shells), but do not follow the international standards. According to their definition, shells do not have to be 2-manifold (an edge is allowed to be incident to more than 2 surfaces), the only criterion is that a solid must have an interior that is connected. Figure 4b shows one example of a valid solid according to their definition, notice that one edge has 4 surfaces incident to it and thus this solid is not valid according the international standards. To be valid, one would have to split the solid into (at least) two solids.

4 A methodology to validate the geometry of a solid

The international standard ISO 19107, as described in Section 2, states that for a three-dimensional primitive to be valid, all its lower-dimensionality primitives should also be valid. The methodology proposed in this paper for the validation of a solid is thus performed in a hierarchical way, starting from the lowest dimensionality primitives. Figure 5 shows the workflow used, with

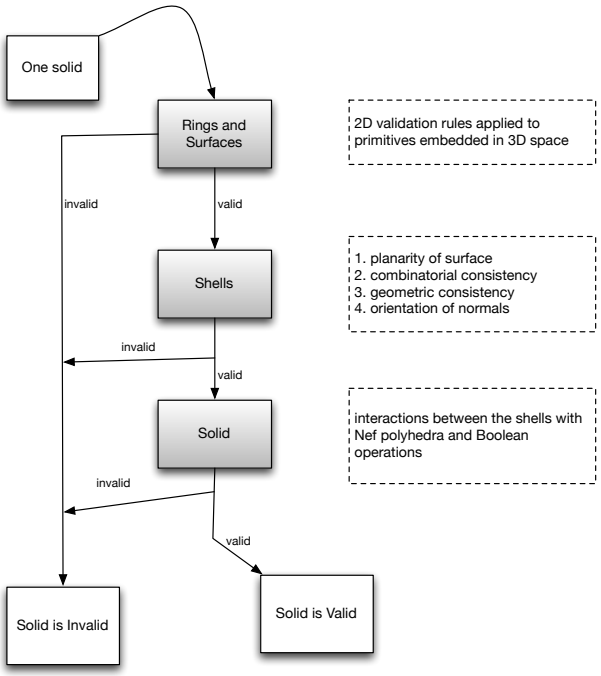


Figure 5: Workflow of the methodology proposed.

some of the verifications that need to be performed.

I describe in this section a methodology to validate each of these primitives. Rings and polygons are validated together with a graph-based data structure since it permits us to represent both the rings and the interactions between them. The validation of a shell is a topic that has been extensively discussed, and I report on the main existing techniques and I highlight pitfalls. The novelty of this paper lies in the Sections 4.3 and 4.5.

4.1 Validation of 2D surfaces with a planar graph

The validation rules for 2D polygons are defined by the ISO [ISO, 2003], but there exists an implementation specification [OGC, 2006]. The problem is considered as solved, although some implementation issues related to the tolerance (shortest distance allowed between points and lines, which is called the *robustness* of a polygon) are still problematic. Van Oosterom et al. [2004] highlight that fact, which has not been solved yet in consistent manner. There exist several implementations of these implementation rules (GEOS and JTS are two open-source examples), these usually build a planar graph of all the rings of a polygon and enforce topological and geometrical constraints. To deal with floating-point arithmetic, these two libraries snap all vertices to an integer grid, which can cause sliver polygons to collapse to lines. Other methods are also known, for instance the use of a constrained triangulation combined with robust arithmetic [Ledoux et al., 2012].

To use the existing algorithms, each surface of a solid will be processed individually and be projected to a plane so that a two-dimensional coordinate system is used. It is possible to find the equation of the plane supporting the surface, but that requires unnecessary and expensive computations. Indeed, if we ensure that a projected polygon does not become a line, then the topology of a polygon is preserved after being projected to a surface. It suffices to project a polygon to either the $x-y$, the $x-y$ or the $y-z$ plane.

4.2 Validation of individual shells with a 2-manifold data structure

The validation of 2-manifold models is a well-studied topic. Most algorithms employ a graph-based data structure to ensure that the topology between the faces is correct and that the surface of the 2-manifold does not have a boundary (in the topological sense of the term) [Kettner, 1999]. The geometric consistency is ensured in another step: pair-wise intersection tests for all the faces

must be performed. Ensuring that a 2-manifold is valid (combinatorially and geometrically) can be done with 13 axioms, as Gröger and Plümer [2011] demonstrate. However, to implement these one must first build a graph representation of the object, and if this object is not valid then it might not be possible (depending on the data structure used). In practice, building incrementally a 2-manifold with Euler operators (which are topological operators) is a better alternative. It is known that orientable 2-manifold models are closed under Euler operations [Mäntylä, 1988]. If the construction of a 2-manifold object has succeeded with a series of Euler operators, the resulting object is not necessarily closed (a surface could have been created). To ensure that the object is closed, it suffices to ensure that each edge has 2 incident faces.

Both Gröger and Plümer [2011] and Mäntylä [1988] assume that the graph representing the 2-manifold object is connected, which is not always the case with ISO solids. Braid et al. [1978] first presented a modification to handle holes and to use Euler operators, but a simpler solution is to triangulate the surfaces having holes. To be triangulated, a polygon must first be projected to a plane; the same method as presented in Section 4.1 can be used. The triangulation itself can be performed with known algorithms for constrained triangulation, see for instance Shewchuk [1997].

Verifying the geometric consistency of a 2-manifold model has a quadratic behaviour. To speed up the process, auxiliary data structures and algorithms have been designed: in Zhou and Suri [1999] and Zomorodian and Edelsbrunner [2002] only surfaces whose bounding boxes overlap are tested for intersection.

I discuss in the following two further requirements that are part of the ISO definition: (1) planarity of surfaces; (2) orientation of surfaces.

Planarity of surfaces. The ISO 19107 standards state that a surface must be planar, i.e. all its vertices (exterior and interior rings) must lie on a plane. Interestingly, the concept of *tolerance* is not mentioned in the standards, nor is any algorithm describing how planarity should be tested. A naïve implementation (without tolerance) with real-world data (using floating-point arithmetic) will almost in all cases return that the face is not planar. Wagner et al. [2012] investigate three different methods to ensure flatness of a surface, and they report that with their test dataset the method that catches the most errors is the one where a triangulation of the surface is performed, and then the orientation of the normal of each triangle is compared. Other methods where a plane is fitted through the points is not robust for small and sharp variations. Since the methodology I propose already triangulates all surfaces, this is the method chosen for the implementation. It should however be noticed that in mechanical engineering there are specific standards to verify the flatness of a surface (see for instance ASME [2009]), and that these use more complex methods—Hossein Cheraghi et al. [1996] and Lee [1997] are two examples.

Orientation of surfaces A surface used to represent a shell must be oriented in such as way that when viewed from outside the shell its vertices are ordered counterclockwise [ISO, 2003]—in other words, the normal of the surface point outwards if a right-hand system is used. If the polygon used to represent the surface has interior rings, then these have to be ordered clockwise. Given a simple surface, it is not possible to define if its orientation is correct since this is an operation requiring a global view on the shell. Therefore, if the previous validation tests are successful then either the normals are all in the correct direction, or all in the wrong direction (if only one surface had a wrong orientation, then the data structure used to validate the topology would be invalid). It suffices to test if the normal (the vector) of one surface incident to the lowest-right vertex intersects another surface. This can be performed efficiently if the auxiliary data structure used for the geometry consistency tests is used. Observe that for interior shells, the orientation of the surfaces must be the opposite: the normals must point inwards (but outwards for the solid).

4.3 Validation of solids with the Nef polyhedron

From a point-set topology point-of-view, a shell is a point set $H \subseteq \mathbb{R}^3$ (i.e. the planar faces of the shell represent ∂H; H is both the boundary of the shell and its interior H^o), and a solid is a point set $S \subseteq \mathbb{R}^3$ such that $S = G \setminus (H_1^o \cup H_2^o \cup \ldots \cup H_n^o)$, where G is the exterior shell, and H_i^o the interior of interior shells (for $i = \{1, 2, \ldots, n\}$).

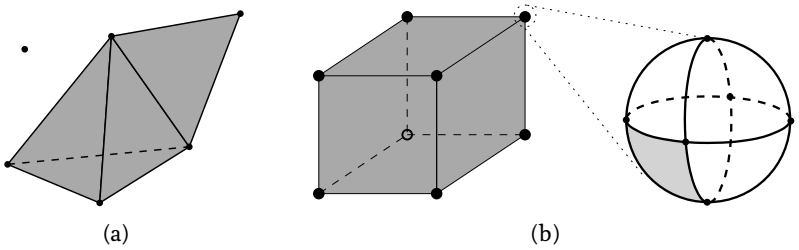


Figure 6: **(a)** A Nef polyhedron composed of a tetrahedron, a dangling triangular face incident to the tetrahedra and an unconnected vertex. **(b)** The sphere map used to store the local neighbourhood of one corner of a cube. The grey part of the sphere represents the interior of the cube, obtained from oriented half-spaces.

If 2 shells overlap or touch, we need to be able to detect it to validate a solid. Simply testing for intersections of the primitives of different dimensionality is not enough since the boundaries of shells are allowed to intersect (solid s_3 in Figure 3 is valid) but if they intersect at several locations the solid might become unconnected (s_9 in Figure 3 is not valid). A global view of the 2 shells is thus needed.

Nef polyhedra. To understand and extract the topological relationships between the different shells forming a solid, I propose using the concept of Nef polyhedra. Nef [1978] and Bieri and Nef [1988] describe a three-dimensional Nef polyhedron as follows.

Definition 4 *A Nef polyhedron in dimension 3 is a point-set $S \subseteq \mathbb{R}^3$ generated from a finite number of open half-spaces by set complement and set intersection operations.*

The fact that they are obtained by intersecting oriented half-spaces implies that they are very general. Unbounded solids, primitives of different dimensionalities (for instance a single point), non-manifold objects, and the empty set are allowed. Figure 6a shows one example, notice that the Nef polyhedron has one dangling surface, and also one unconnected vertex forms the polyhedron. Each half-space is oriented: one side is labelled as *IN* and the other as *OUT*. Also, the Nef polyhedra are solids closed under Boolean operations. Observe that it is not possible to use 2-manifold objects to perform a series of Boolean operations since the result of one operation can be one or several non-manifold objects.

An important concept is that of a *face*, which is defined as a maximal subset of $\mathbb{R}^3$ such that all its points have the same *local neighbourhood*. Based on that concept, we can partition a Nef polyhedron into faces of different dimensionalities: a 0-face is a vertex, a 1-face is an edge, a 2-face is a planar surface, and a 3-face is a *volume* (defined as a connected point-set in $\mathbb{R}^3$).

The generality of Nef polyhedra is not *per se* needed for the validation of ISO solids. However, I exploit two of their characteristics to analyse the topological relationships between shells. First, non-manifold objects can be represented, and second, we can construct and manipulate Nef polyhedra with a series of Boolean operations. It should be stressed that validating solids requires representing and storing invalid solids too. In brief, the generality of the Nef polyhedra permits us to identify invalid solids, which is the purpose of this paper.

4.4 Storing and manipulating Nef polyhedra

To perform Boolean operations, we first need to represent a Nef polyhedron with an appropriate data structure. Bieri and Nef [1988] describe a potential data structure, but it does not support Boolean operations. Granados et al. [2003] improve on that structure to support Boolean operations. For the methodology presented in this paper, I make use of this structure (without any modifications). What follows is a summary of the data structures used by Granados et al. [2003], for all the details the reader is referred to the original paper.

While a Nef polyhedron is a three-dimensional object that is potentially non-manifold, Granados et al. [2003] represent one with a set of 2D data structures. Indeed, two different data structures are used, both of these being similar to an edge-based data structure for 2-manifold objects. The two data structures are inter-connected, and are as follows.

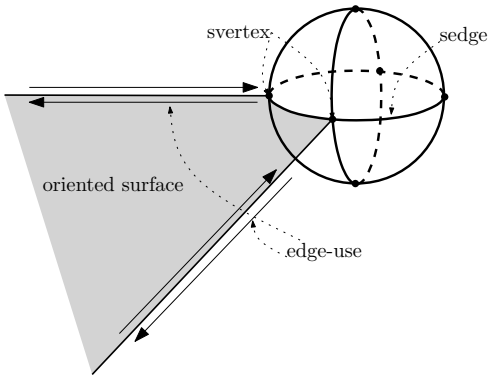


Figure 7: The two interconnected data structures, used to store Nef polyhedra, for the top surface of the sphere map shown in Figure 6b. Figure after Granados et al. [2003].

Sphere maps: the local neighbourhood of a given point in a Nef polyhedron is represented with an infinitesimally small sphere called a sphere map. Figure 6b shows an example for one vertex of a cube. Notice that each oriented half-space partitions the boundary of the sphere and creates an edge (called a *sedge*, it is a great circle if only one half-space intersects a sphere map); vertices on the sphere are *svertices* representing the intersection of at least two half-spaces (thus an edge in a Nef polyhedron); faces (*sfaces*) of the sphere represent volumes in the Nef polyhedron. As is the case for half-spaces, each *sface* will have either an *IN* or an *OUT* label to keep track of the interior of the Nef polyhedron. The partitioning of the sphere is stored with an edge-based structure similar to the half-edge [Mäntylä, 1988]. Not all the points of $\mathbb{R}^3$ need to be represented with a sphere map, but only the vertices of the Nef polyhedron. This permits us to detect non-manifold cases of vertices lying directly on another face, as s_3 in Figure 3 shows (the apex of one interior shell is coplanar with one side surface of the exterior shell). Other non-manifold data structures such as the radial-edge [Weiler, 1988] encode non-manifold situations for edges, but would need modifications to explicitly represent the cases arising at vertices (and thus cannot be used directly for the validation of solids).

b-rep for surfaces: another data structure, also inspired by the half-edge, is used for the surfaces, and that structure is linked to the incident sphere maps. Each half-edge belonging to one orientation of a surface (an *edge-use*) is stored, and is linked to the incident edge-uses on the face. It is furthermore linked to the svertex of the sphere map it represents, and vice-versa.

Figure 7 shows the two data structures for one 2-face of a Nef polyhedron.

A Boolean operation is performed by either updating the sphere maps, or by creating new ones at the locations of the intersections of half-spaces.

These two inter-connected data structures permit us to navigate and keep track of the primitives (including the connected parts), which allows us to identify, for instance, the fact that solid s_9 in Figure 3 contains more than one volume. The incident surfaces of a given edge can for instance be obtained by identifying the incident sedges to the svertex that the edge represents. And the sphere maps allow us to navigate to all the primitives around a vertex, even if it is non-manifold.

The unconnected components of a Nef polyhedron are maintained with a kd-tree, which permits fast point location (to detect that one volume is located inside another one for instance).

4.5 Boolean operations to analyse the topological relationships between shells

At this point of the validation each shell is individually valid. The invalid solid of Figure 4b would therefore not be processed at this step because the validation rules of Section 4.2 would fail.

The algorithm to extract the topological relationships between the different shells of a solid is based on the fact that we can navigate in a Nef polyhedron and report on its primitives (the two

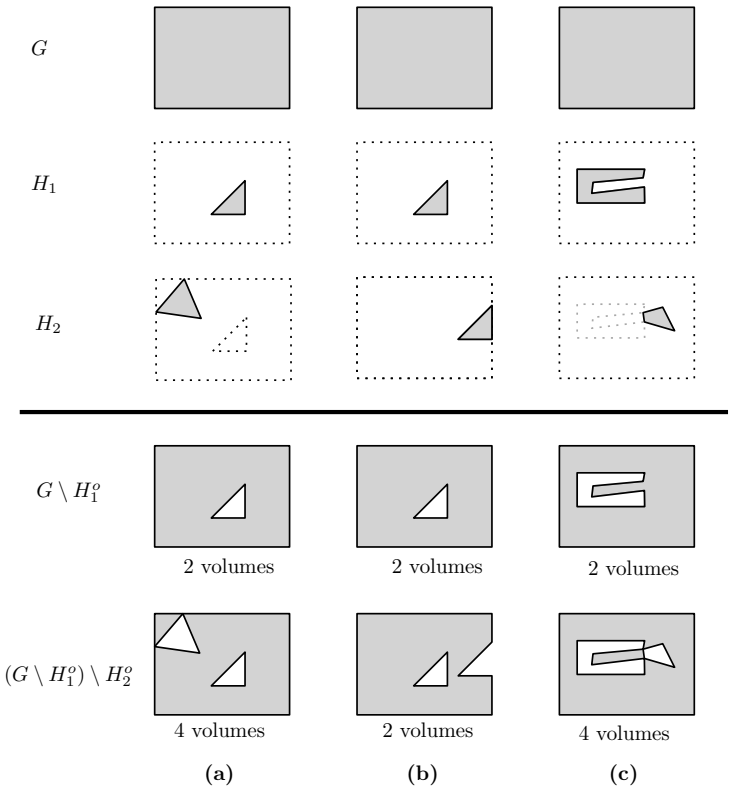


Figure 8: Three invalid solids formed by an exterior shell G and two interior shell(s) (H_1 and H_2). The resulting number of volumes in the Nef polyhedron for each $G \setminus H_i^o$ is shown. **(a)** three of the vertices of ∂H_2 intersect ∂G, thus the interior of the solid is unconnected. **(b)** H_2 is face adjacent with G ($\partial G \cap \partial H_2$ returns a triangle surface). **(c)** H_2 makes the interior of the solid unconnected (as in **(a)**).

inter-connected data structures permit that). It is based on a series of Boolean operations, which are expressed with the following three axioms. G is the exterior shell of a solid S containing n interior shells, and H_i is an interior shell. The operator $dim()$ returns the maximum dimensionality of a set of primitives.

1. $G' \cap H_i = \emptyset$
2. $dim(H_i \cap H_j) < 2$
3. $S_i = G \setminus (H_1^o \cup H_2^o \cup ... \cup H_i^o)$, for each i in $\{1, 2, ..., n\}$, contains exactly $i + 1$ volumes.

The first axiom ensures that no interior shell is located outside, or partially outside, the exterior shell. The second axiom ensures that the intersection of two interior shells returns only primitives of dimensionality 0 or 1—as described in Section 2 two interior shells can only 'touch' along vertices and edges. The third axiom ensures that the intersections between the different shells respect the ISO definition of a solid, and that they do not create, among others, a disconnected volume. Verifying the third axiom involves doing a set difference on G and each interior shell H_i incrementally, ensuring that the number of volumes is increased by 1 at each step. Figure 8 shows the idea for three solids; the figure is in two dimensions for the sake of clarity, but since Nef polyhedra can be manipulated using topological operators the result is the same. In three dimensions, a square becomes a cube, and a triangle a tetrahedron. The number of shells determines the number of volumes that the solid should have, according to assertion 6 in Figure 2. Notice that none of the solids have the correct amount of volumes, and thus none are valid. Observe also that it is not necessary to have an axiom verifying if G and one shell H_i have adjacent faces, since the third axiom implicitly does that.

5 A robust C++ implementation using CGAL

The methodology described in this paper has been implemented in a prototype, called `val3dity`[6], that is freely available under a GPL-license. The code is written in C++ and is mostly based on the CGAL library[7]. The only other library used is GEOS[8] which performs the two-dimensional validation with the approach described in Section 4.1.

CGAL was chosen because it contains several of the building blocks required to implement a validator for ISO solids: (i) a package for constructing shells and representing them with the half-edge data structure; (ii) the algorithm of Zomorodian and Edelsbrunner [2002] to avoid a quadratic behaviour when performing the geometric tests for the shells is implemented; (iii) a constrained triangulator [Boissonnat et al., 2002]; (iv) and the Nef polyhedra data structures and Boolean operators described in Granados et al. [2003]. Because the building blocks are available, the code for the validation is rather short: about 2500 lines of code, which makes it easy to maintain and modify.

I describe briefly in this section some of the engineering choices that were made for the development of the prototype. Some were made to ensure that the prototype runs correctly and efficiently, and some to provide meaningful feedback to the users of the prototype. A trade-off between speed and the granularity of the error returned to the user had to be made.

5.1 Robustness

The implementation of geometric algorithms is known to be a time-consuming and error-prone task because of the several degeneracies arising and because of floating-point arithmetic, especially for three-dimensional operations [Hoffmann, 1989]. CGAL offers the possibility to use *exact* arithmetic [Yap and Dubé, 1995] for all the packages and `val3dity` makes use of it to ensure that it is robust. All the important steps make use of it: the constrained triangulator and the polyhedra use robust predicates, and the Nef polyhedra use a robust kernel for all the Boolean operations. Hachenberger et al. [2007] show that the Nef polyhedra implementation in CGAL is in practice fast (comparable speed as a commercial kernel), but it is also more robust. Filtering of the floating-point numbers is performed so that exact computations only take place when needed [Pion and Fabri, 2011].

5.2 Errors returned to the user

The validator for solids in Oracle Spatial permits us to validate solids (although, as explained in Section 3 it is neither according to the ISO rules nor complete) but returns only one error when the solid is not valid: the first one encountered (even if a given solid contains hundreds of errors). The error comes with a code explaining its nature and, when suitable, its location (for example if a shell is not closed the centre of the hole is given). This means that a user has to fix the solid for the error mentioned, and to run again the validation function. This step has to be followed for all the errors present, which can be a rather long and painful process for the user.

Ideally, all the errors in a solid should be reported so that a user can fix them in one operation. However, cascading effects when validating should be avoid—one example is if a surface is not a valid polygon in 2D, then the validation of the shell whose boundary contains that surface should not be attempted as it will most likely not be valid. In the prototype `val3dity`, a "hierarchical validation" is used and efforts are made to avoid cascading errors. As Figure 9 shows, first the surfaces of every shell are validated, if all of them are valid then the shells are validated individually, and finally only when all the shells of a solid are valid is the validation of the solid with the Nef polyhedron data structure performed.

Informing the user about the nature of the error and having the most efficient code are contradictory goals. In `val3dity`, a trade-off has been made for the implementation. For instance, it is relatively easy to report that a given shell is not a 2-manifold object if a batch construction is performed and exited as soon as an error is encountered. Reporting on the exact error often makes the process slower (and more cumbersome to program); it was nevertheless chosen for the

[6] `https://github.com/tudelft3d/val3dity`

[7] Computational Geometry Algorithms Library: `http://www.cgal.org`

[8] Geometry Engine—Open Source: `http://trac.osgeo.org/geos/`

```
Surface level
=============
200   : exterior ring and interior rings have same orientation
210   : surface is not planar
220   : surface is not valid in 2D (its projection)
  221 : interior ring intersect exterior ring
  222 : interior ring outside exterior ring
  223 : interior rings intersect
  224 : interior not connected

Shell level
===========
300   : is not a 2-manifold
  301 : surface is not closed
  302 : dangling faces
  303 : one(several) face(s) not connected to the 2-manifold
  304 : orientation of one/several face(s) not correct
  305 : surface self-intersect
310   : normals not pointing in correct direction (all of them)

Solid level
==========
400   : shells are face adjacent
410   : interior of shells intersect
420   : interior shell outside the exterior shell
430   : interior not connected
```

Figure 9: Error codes used in the prototype.

prototype as it was deemed more useful for the user. In this case, it means that if a dangling face exists (let us assume there is only one), then the identifier of the face in the input dataset should be reported. An incremental construction of the shell should therefore be used, with verification of the validity at each step.

6 Experiments with real-world datasets

The prototype has been tested with different datasets, both real-world and synthetic. It takes as input a set of shells, the first one being the exterior one and the others the interior shells. Each shell is represented with a simple ASCII file following the format POLY[9], which can be seen as a generalisation of a simpler format for representing a simple polyhedron—this is needed since holes in surfaces must be explicitly stored.

"Unit test" solids. Figure 3 contains some of the unit test solids that were tested. These are purposefully constructed solids that represent extreme and special cases. They are meant as a sort of *unit testing* solids to evaluate the methodology and its implementation [Burns, 2001]. All the solids shown in the figure, and other similar ones, have been used during the development and are validated correctly.

3D city models. City models are often stored in the CityGML format [OGC, 2012], which uses GML [OGC, 2007, the Geography Markup Language][10]. This representation does not store any topological relationships between the surfaces forming a shell, i.e. a cube is represented with 6 faces and for each one of these the coordinates of the 4 vertices are enumerated. Snapping must thus be performed to extract unique vertices in one shell; a user-defined tolerance must be used. To validate the solids in a city model, they are first converted to POLY files (where vertices have labels and faces of shells are represented by enumerating labels), and then validated.

One experiment was performed with a test dataset available on the CityGML webpage[11], it is from the UK's Ordnance Survey and it contains 567 buildings, all obtained by extrusion. Figure 10a shows an overview of the dataset. A snapping of 0.01m was used, and each building was validated independently. Probably because the algorithm/implementation used to construct the dataset contained errors, none of the buildings are valid. Figure 10b and Figure 10c show the

[9] http://tetgen.berlios.de/fformats.poly.html

[10] It should be noticed that CityGML does not use the latest version, but version 3.1.1.

[11] http://www.citygml.org

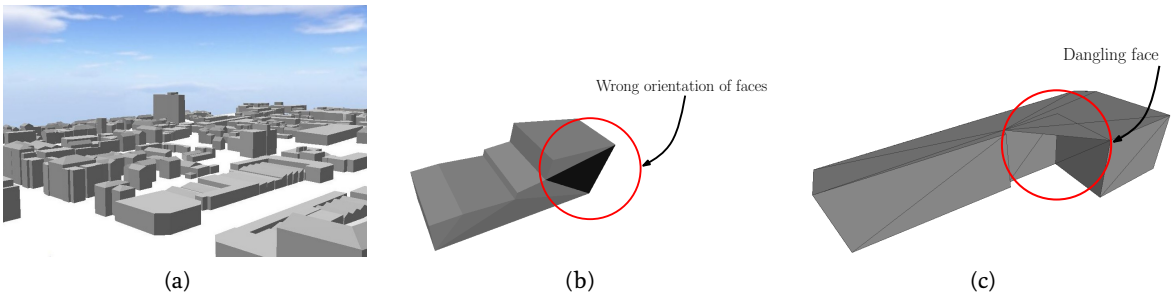


Figure 10: **(a)** Overview of a CityGML test dataset, in Great Britain. (Figure from `http://www.citygml.org`) **(b–c)** Two examples of validation errors in the dataset.

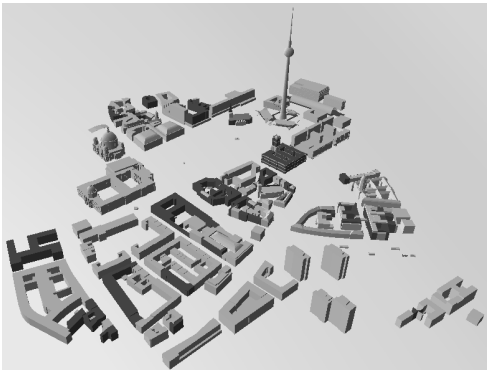


Figure 11: The CityGML data "Berlin Alexanderplatz", the darker solids are invalid.

two most common errors: surfaces (which were all triangulated by the provider of the dataset) do not have the correct orientation, and some shells contain dangling surfaces (probably caused by a faulty triangulator for non-convex surfaces). Another error that was present in the dataset: surfaces were not planar.

Another experiment was performed with the dataset "Berlin Alexanderplatz" available from the same website, it contains 1123 solids, 237 of them being invalid. The errors present were (from Figure 9): 210, 301, 302, 304 and 305. The prototype `val3dity` can also return a CityGML file where the invalid solids are highlighted, with the error code for each solid, as Figure 11 shows.

Larger datasets. Most of the solids in real-world datasets obtained by extrusion have very few surfaces (usually less than 30), so experiments were made with larger dataset. One example is shown in Figure 12 it contains 1696 faces (triangles), and 836 vertices. The validation was performed in about 1.5s on a standard laptop.

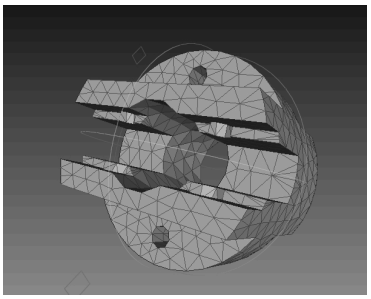


Figure 12: A mechanical piece, already triangulated.

Other software. For the reasons elaborated in Section 3, other tools could not be tested in a meaningful manner with the previous datasets. Some would be suitable (e.g. for the simple extrusion dataset of the UK's Ordnance Survey), but if holes and cavities were present, these would not be able to handle them.

7 Discussion and conclusions

Having validation tools that respect the international standards help foster interoperability and help applications where city models form the input to models. Examples are flood modelling [Schulte and Coors, 2008], 3D navigation [Lee and Zlatanova, 2008], disaster management [Kolbe et al., 2008] and urban planning [Köninger and Bartel, 1998]. Having invalid primitives could prevent these models from running properly.

While for years there have been implementation specifications and tools to validate two-dimensional primitives, three-dimensional volumetric primitives have been represented and stored in databases according to different rules and definitions. Particularly problematic is the fact that holes in surfaces and cavities are ignored, while they are allowed according to the ISO/OGC abstract specifications and there exists several datasets containing them. It is expected that more and more datasets having holes and cavities will become available in the near future. Two concrete examples are the interior of buildings obtained from IFC models [Lee et al., 2003] and geological datasets [Adams, 1994].

The results of this paper are two-fold:

1. the implementation specifications for 3D geometric primitives have been proposed, and concrete examples of these have been shown. The main issue with solids represented with the ISO/OGC standards is that cavities require the use of non-manifold data structures.

2. a methodology for validating solids has been proposed, and I have shown that it is possible to implement it in a robust and efficient manner. The prototype implementation reuses many of the building blocks developed in other research about the representation and the validation of 3D data, but goes one step further as solids having interior shells are considered, which is consistent with the international standards ISO 19107 [ISO, 2003]. As the developed prototype is available under an open-source license, I hope that it will encourage software vendors and others to adhere to the international standards.

The methodology, and its implementation, presented in this paper permits us to verify that an *individual* solid respects the ISO/OGC definition. However, it does not permit us to detect that two solids touch or overlap each other. That is useful in different applications where the topological consistency of a whole city model is needed, for instance for analysing the energy consumption of buildings the area shared between buildings is one factor to model [Krüger and Kolbe, 2012]. Ledoux and Meijers [2011] propose a methodology to *construct* by extrusion such a dataset from two-dimensional footprints, but I am not aware of any methodology for already available datasets. One potential solution would be to use Nef polyhedra (and Boolean operations), but that would be costly for big datasets.

At this moment, only the geometry and the topology of a solid is considered when validating. However, as Bogdahn and Coors [2010] state, validation rules are often application-specific. Wagner et al. [2012], for the modelling of 3D buildings, discuss the use of semantics information when validating, i.e. if for instance one surface is labelled as the roof of the building, then an extra validation rule (over the geometry) would be to ensure that the roof is located 'above' the surface labelled as the ground floor. Since the methodology I present in this paper is hierarchical and it uses data structures supporting Boolean operations, the implementation of similar rules should be straightforward.

Finally, one obvious extension of the work presented in this paper is the *automatic* repair of invalid solids. At this moment, the prototype informs the user—as best as it can—about the nature of the errors and of their locations, but the user has to manually modify the faulty primitives, which is tedious and time-consuming. The automatic filling of holes in 2-manifold objects is a well-studied topic of research, and the existing techniques developed could be reused. Nooruddin and Turk [2003] fill the object with voxels, but this method could introduce errors depending on the resolution used. Liepa [2003] and Attene and Falcidieno [2006] attempt the same by modifying the triangles on the surface; their approach require a triangulated surface but this is

inline with the methodology presented in this paper. Other useful automatic repair functions for shells are the flipping of surface when their orientation are wrong, and the triangulation of a non-planar surface so that only planar surfaces exists. Shells having several holes and intersecting surfaces are more problematic to repair, although work done in surface reconstruction could be used, see for instance Chauve et al. [2010]. To automatically repair solids whose shells do not interact according to the international standards is in theory possible since with the Nef polyhedra we can perform Boolean operations. However, how to repair is also application-dependent: if for instance one solid has an interior shell located *outside* its exterior shell, what should be the outcome of a repair operation? Two solids or only the exterior solid? When two or more interior shells overlap, it is however trivial to union them and return one interior shell.

Acknowledgements

This research was financially supported by Safe Software Inc., the Dutch Technology Foundation STW (project code: 11300), and by the "3D Pilot" in the Netherlands (thank you Jantien Stoter for giving validation an important role). I would like to thank Junqiao Zhao, Ken Arroyo Ohori, Kevin Wiebe, Brittany Zenger and Jan Kooijman for helping out with the implementation and for very useful discussions.

References

Adams TM (1994). GIS-based subsurface data management. *Computer-Aided Civil and Infrastructure Engineering*, 9(4):305–313.

ASME (2009). Y14.5-2009: Dimensioning and tolerancing. American Society of Mechanical Engineers.

Attene M and Falcidieno B (2006). ReMESH: An interactive environment to edit and repair triangle meshes. In *Proceedings IEEE International Conference on Shape Modeling and Applications*, pages 271–276. Matsushima, Japan.

Bieri H and Nef W (1988). Elementary set operations with d-dimensional polyhedra. In *Proceedings on International Workshop on Computational Geometry on Computational Geometry and its Applications*, pages 97–112. Springer-Verlag New York, Inc., Würzburg, Germany.

Bogdahn J and Coors V (2010). Towards an automated healing of 3D urban models. In Kolbe TH, Köning G, and Nagel C, editors, *Proceedings International Conference on 3D Geoinformation*, volume ISPRS Volume XXXVIII-4/W15, pages 13–17.

Boissonnat JD, Devillers O, Pion S, Teillaud M, and Yvinec M (2002). Triangulations in CGAL. *Computational Geometry—Theory and Applications*, 22:5–19.

Braid IC, Hillyard RC, and Stroud IA (1978). Stepwise construction of polyhedra in geometric modeling. Technical Report CAD Group Document No. 100, Computer Laboratory, University of Cambridge.

Burns T (2001). Effective unit testing. *ACM Ubiquity*, 2001, issue January(Article no. 1).

Chauve AL, Labatut P, and Pons JP (2010). Robust piecewise-planar 3D reconstruction and completion from large-scale unstructured point data. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, pages 1261–1268.

de Berg M, van Kreveld M, Overmars M, and Schwarzkopf O (2000). *Computational geometry: Algorithms and applications*. Springer-Verlag, Berlin, second edition.

Fortune S (1997). Polyhedral modelling with multiprecision integer arithmetic. *Computer-Aided Design*, 29(2):123–133.

Granados M, Hachenberger P, Hert S, Kettner L, Mehlhorn K, and Seel M (2003). Boolean operations on 3D selective Nef complexes: Data structure, algorithms, and implementation. In *Proceedings 11th Annual European Symposium on Algorithms (ESA'03)*, number 2832 in Lecture Notes in Computer Science, pages 654–666. Budapest, Hungary.

Gröger G and Plümer L (2011). How to achieve consistency for 3D city models. *GeoInformatica*, 15:137–165.

Grünbaum B (2003). Are your polyhedra the same as my polyhedra? In Aronov B, Basu S, Pach J, and Sharir M, editors, *Discrete and Computational Geometry: The Goodman-Pollack Festschrift*, pages 461–488. Springer.

Guibas LJ and Stolfi J (1985). Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123.

Hachenberger P, Kettner L, and Mehlhorn K (2007). Boolean operations on 3D selective Nef complexes: Data structure, algorithms, optimized implementation and experiments. *Computational Geometry*, 38(1–2):64–99. doi:DOI:10.1016/j.comgeo.2006.11.009. Special Issue on CGAL.

Hoffmann CM (1989). The problems of accuracy and robustness in geometric computation. *Computer—IEEE Computer Society Press*, 22:31–42.

Hossein Cheraghi S, Lim HS, and Motavalli S (1996). Straightness and flatness tolerance evaluation: an optimization approach. *Precision Engineering*, 18(1):30–37.

ISO (2003). ISO 19107:2003: Geographic information—Spatial schema. International Organization for Standardization.

Kazar BM, Kothuri R, van Oosterom P, and Ravada S (2008). On valid and invalid three-dimensional geometries. In van Oosterom P, Zlatanova S, Penninga F, and Fendel E, editors, *Advances in 3D Geoinformation Systems*, Lectures Notes in Geoinformation and Cartography, chapter 2, pages 19–46. Springer Berlin Heidelberg.

Kettner L (1999). Using generic programming for designing a data structure for polyhedral surfaces. *Computational Geometry—Theory and Applications*, 13:65–90.

Kolbe TH, Gröger G, and Plümer L (2008). CityGML—3D city models and their potential for emergency response. In Zlatanova S and Li J, editors, *Geospatial Information Technology for Emergency Response*, ISPRS Book Series, pages 257–274. Taylor & Francis, London.

Köninger A and Bartel S (1998). 3d-GIS for urban purposes. *Geoinformatica*, 2(1):79–103. ISSN 1384-6175.

Krüger A and Kolbe TH (2012). Building analysis for urban energy planning using key indicators on virtual 3D city models—the energy atlas of Berlin. In *Proceedings XXII ISPRS Congress*, volume XXXIX-B2, pages 145–150. Melbourne, Australia.

Ledoux H, Arroyo Ohori K, and Meijers M (2012). Automatically repairing invalid polygons with a constrained triangulation. In Gensel J, Josselin D, and Vandenbroucke D, editors, *Proceedings 15th AGILE International Conference on Geographic Information Science*. ISBN 978-3-642-29062-6.

Ledoux H and Meijers M (2011). Topologically consistent 3D city models obtained by extrusion. *International Journal of Geographical Information Science*, 25(4):557–574.

Ledoux H, Verbree E, and Si H (2009). Geometric validation of GML solids with the constrained Delaunay tetrahedralization. In De Maeyer P, Neutens T, and De Ryck M, editors, *Proceedings 4th International Workshop on 3D Geo-Information*, pages 143–148.

Lee J and Zlatanova S (2008). A 3D data model and topological analyses for emergency response in urban areas. In Zlatanova S and Li J, editors, *Geospatial Information Technology for Emergency Response*, ISPRS Book Series, pages 143–168. Taylor & Francis, London.

Lee K, Chin S, and Kim J (2003). A core system for design information management using industry foundation classes. *Computer-Aided Civil and Infrastructure Engineering*, 18(4):286–298.

Lee MK (1997). A new convex-hull based approach to evaluating flatness tolerance. *Computer-Aided Design*, 29(12):861–868.

Liepa P (2003). Filling holes in meshes. In *Proceedings 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 200–205.

Mäntylä M (1988). *An introduction to solid modeling*. Computer Science Press, New York, USA.

Muller DE and Preparata FP (1978). Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7:217–236.

Nef W (1978). *Beiträge zur Theorie der Polyeder*. Herbert Lang, Bern.

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

OGC (2006). OpenGIS implementation specification for geographic information—simple feature access. Open Geospatial Consortium inc. Document 06-103r3.

OGC (2007). Geography markup language (GML) encoding standard. Open Geospatial Consortium inc. Document 07-036, version 3.2.1.

OGC (2012). OGC city geography markup language (CityGML) encoding standard. Open Geospatial Consortium inc. Document 12-019, version 2.0.0.

Oracle (2012). Oracle database 11g documentation.

Pion S and Fabri A (2011). A generic lazy evaluation scheme for exact geometric computations. *Science of Computer Programming*, 76(4):307–323.

Schulte C and Coors V (2008). Development of a CityGML ADE for dynamic 3D flood information. In *Proceedings Joint ISCRAM-CHINA and GI4DM Conference on Information Systems for Crisis Management*. Harbin, China.

Shewchuk JR (1997). *Delaunay Refinement Mesh Generation*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburg, USA.

Thompson RJ and van Oosterom P (2011). Axiomatic definition of valid 3D parcels, potentially in a space partitions. In van Oosterom P, Fendel E, Stoter J, and Streilein A, editors, *Proceedings 2nd International Workshop on 3D Cadastres*, pages 397–416. Delft, the Netherlands.

van Oosterom P, Quak W, and Tijssen T (2004). About invalid, valid and clean polygons. In Fisher PF, editor, *Developments in Spatial Data Handling—11th International Symposium on Spatial Data Handling*, pages 1–16. Springer.

Verbree E and Si H (2008). Validation and storage of polyhedra through constrained Delaunay tetrahedralization. In Cova TJ, Miller HJ, Beard K, Frank AU, and Goodchild MF, editors, *Proceedings 5th international conference on Geographic Information Science*, volume 5266 of *Lecture Notes in Computer Science*, pages 354–369. Springer.

Wagner D, Wewetzer M, Bogdahn J, Alam N, Pries M, and Coors V (2012). Geometric-semantical consistency validation of CityGML models. In Pouliot J, Daniel S, Hubert F, and Zamyadi A, editors, *Lecture Notes in Geoinformation and Cartography*, pages 171–192. Springer Science.

Weiler K (1988). The Radial Edge Structure: A topological representation for nonmanifold geometric boundary modeling. In *Geometric Modeling for CAD Applications*. Elsevier Science, Amsterdam.

Yap CK and Dubé T (1995). The exact computation paradigm. In Du DZ and Hwang FK, editors, *Computing in Euclidean Geometry*, pages 452–486. World Scientific Press, Singapore, second edition.

Zhou Y and Suri S (1999). Analysis of a bounding box heuristic for object intersection. In *Proceedings 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 830–839. Baltimore, Maryland, USA.

Zomorodian A and Edelsbrunner H (2002). Fast software for box intersection. *International Journal of Computational Geometry and Applications*, 12(1–2):143–172.