Are your IndoorGML files valid?

Hugo Ledoux

IndoorGML datasets allow us to represent both (1) the geometry of the interior of a building, which is subdivided into *cells* (eg rooms, corridors, staircases); and (2) the navigation graph between these cells, which also acts as a mechanism to store the topological relationships between the cells. To be used in applications such as indoor routing or emergency evacuation, IndoorGML files should be valid and structured according to the specifications of the OGC. In practice, achieving this is challenging because two different representations of the indoor space must be modelled and linked together; other 3D formats usually only store one representation. In this paper, I present a new methodology to validate IndoorGML files. It builds upon previous work on the validation of 3D geometries and city models, and it contains six specific tests. These tests have been implemented in the open source software *val3dity*, and I present and discuss experiments I ran with all the publicly available IndoorGML datasets I could find.

1 Introduction

IndoorGML is a data model to represent the geometry and semantics of the interior of a building (a building is for instance represented by a set of rooms, corridors, staircase, etc.), and the navigation graph inside that building [OGC, 2014]. As further explained in Section 2, it is standardised by the "Open" Geospatial Consortium® (OGC) and has one standardised implementation (XML-based). It was developed to unify different standards for indoor maps, by incorporating different properties from other standards [Kang and Li, 2017]. The main goal was to have a format focused on navigation applications, but that could also be useful for a wide-range of use-cases, a few examples are emergency evacuation [Hashemi, 2018], indoor routing and distance computation inside buildings [Goetz and Zipf, 2011; Diakité and Zlatanova, 2017; Park et al., 2018], and building accessibility auditing [Dao and Thill, 2017].

To be useful in different applications and be processed by different software, IndoorGML files should be *valid*, ie they should be formatted according to the OGC specifications [OGC, 2014], the geometries should be free of errors such as self-intersections or overlapping solids, and the navigation graph should be consistent with the geometry of the building. Notice that ensuring validity goes (far) beyond the typical validation against the XML schemas that is carried out by certain software. As Section 3.1 explains, many OGC specifications/rules cannot be encoded in XML schemas, and as a consequence, specific code needs to be written.

We know that in practice files representing the 3D geometries of cities are plagued with errors, see among others Biljecki et al. [2016], Steuer et al. [2015], Alam et al. [2014], Mulder [2015], and Pédrinis et al. [2015]. The question tackled in this paper is whether this is also the case with 3D representations of the interior of buildings.

I investigate in this paper which validation rules are necessary to ensure that an IndoorGML model is "error-free" and therefore can be further processed in downstream applications by different software. While I could not find papers directly addressing this issue, much has been written on a very similar topic: the validation of 3D city models. The work presented in this paper reuses and extends methods and ideas from those. It can be seen as extending the standard algorithms for validation of 3D geographic models [Gröger and Plümer, 2011; Wagner et al., 2015; Colley et al., 2017] to specific cases; among others, Coors et al. [2020] did this for the case of heating demand simulation, and Ledoux [2018] for complex buildings having parts (where the topological relationships between parts are validated). I report in Section 3 on the methodology that I have developed, it uses the framework described in Ledoux [2013] for validating the 3D geometries against the definitions in ISO19107 [ISO, 2003], and it builds upon the specific rules described in Ledoux [2018] for CityGML buildings to define new specific rules adapted to the characteristics of indoor models. The main difficulty in designing validation rules for IndoorGML is that, as explained in Section 2, two models of the same building must be validated (primal and dual), and these two should be consistent with each other. This is different from the validation of 3D primitives in 3D GIS. It should be observed that I focus solely on the 3D representation of objects in IndoorGML; 2D representations are possible but are less frequent, and the validation of 2D geometries and graphs has already several implementations, see for instance Davis [2003] and GEOS[1].

As explained in Section 4, I have implemented the proposed validation rules in C++ in the software *val3dity*, an open-source software to validate 3D primitives according to the international definitions of ISO19107. The updated version of the software can read natively IndoorGML files, a series of validation tests are performed, and a summary report is returned to the user. I report on experiments I ran: I validated all the IndoorGML datasets that are publicly available, and some that were given to me. The experiments highlighted that IndoorGML files, like many other datasets in 3D GIS, are riddled with errors. Some of them are easy to fix (eg duplicate vertices), while some others are more complex (eg overlapping rooms in a building).

2 IndoorGML overview

IndoorGML is both a data model and an XML-based exchanged format (both having the same name!) for modelling indoor spaces, with an emphasis on navigation applications. It is standardised by the OGC, v1.0 is the current one, and the latest XML schemas (.xsd) are v1.0.3.

The main idea behind IndoorGML is to semantically decompose the indoor space of a building into *cells*, which are deemed as the simplest unit useful for navigation. Examples of cells are

[1]https://trac.osgeo.org/geos/

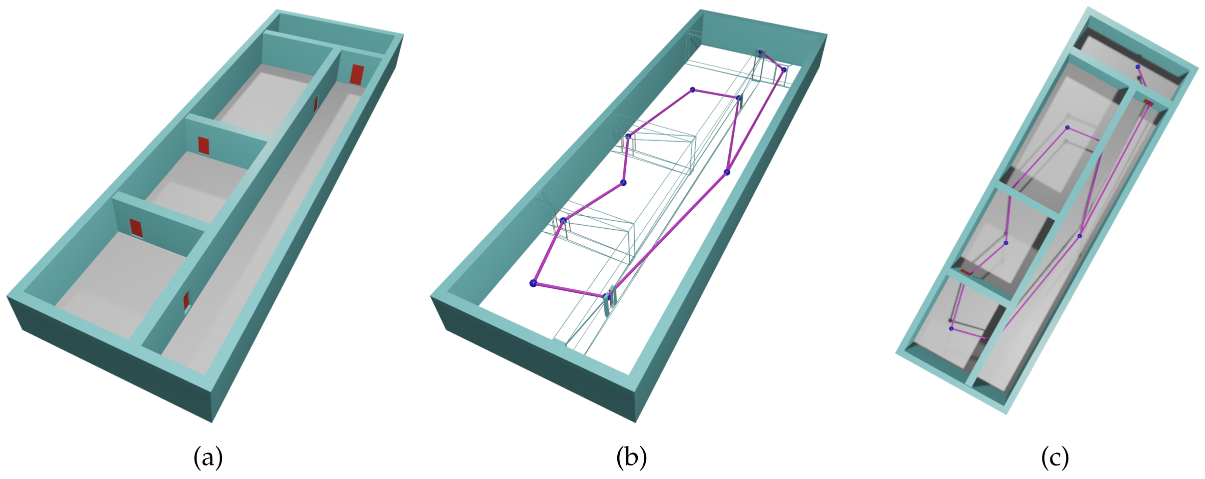


Figure 1: **(a)** The indoor space subdivided into 4 rooms and one corridor, and 5 doors in red. **(b–c)** The navigation graph in purple.

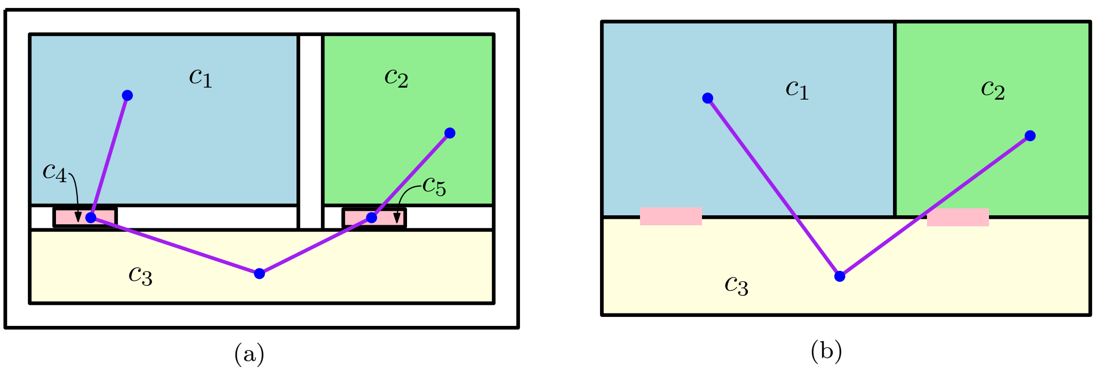


Figure 2: Two ways to model cells in an IndoorGML model. **(a)** 5 cells are present since doors (pink cells) are explicitly modelled. **(b)** The doors are not cells, their presence can be inferred by the fact that the navigation graph has an edge between 2 cells.

rooms, corridors, staircases, but also doors; architectural components such as cornices, windows, and furnitures are usually out-of-scope for IndoorGML datasets. In Figure 1a, the cells are: the 4 rooms, the 5 doors, and the corridor. The geometry of one such cell has to be represented with an ISO19107 `GM_Solid` in 3D (and a `GM_Surface` in 2D, but we ignore this case for this paper), and as further explained in Section 3.2, the geometric primitives are linear/planar in practice. Each cell can have specific attributes and semantics, the latter may come from *extensions*. For v1.0, only one extension is officially supported ("Navigation", which can label corridors as "connection space"), but there are a few in development, eg "Public Safety Features (PSExt)" and "Textured Surface", see `http://indoorgml.net/resources/` for the overview.

The decomposition of the 3D space into cells is supplemented by a navigation graph. This graph has one node per cell, and an edge connects two nodes if the two corresponding cells are adjacent and can be reached directly from one another (with a door for example). Two adjacent rooms not having a door in their shared wall would not have an edge linking their two nodes in the navigation graph.

There are two ways to partition an indoor space into cells, Figure 2 shows them. The first one (Figure 2a) is when cells are defined as the interior of rooms/corridors, walls are omitted, and doors becomes cells (with a thickness) and thus also get a vertex in the navigation graph. The space in Figure 1b–c is modelled this way. The other option (Figure 2b) is to ignore walls, and a cell contains part of the thickness of the walls; in this case the navigation graph only links cells where there is a door (but this door does not need to be explicitly modelled).

The IndoorGML navigation graph does not have to be connected, for example it is possible that one room is accessible only with a door to the outside, and thus it is not connected to the other rooms in the building.

A final property of IndoorGML is that different decompositions of the indoor space and/or different navigation graphs can be stored in the same model. One could think of a navigation graph for pedestrians, and one for people in a wheelchair, the constraints of the latter would make the graph having fewer edges.

Notice that in the IndoorGML specifications, the nodes of the navigation graph are called "States" and the edges "Transitions", but as argued in Alattas et al. [2018], this is confusing and the more common terms "nodes" and "edges" are used in the following.

The decomposition of the 3D space of an indoor model into cells can be conceptualised as a *cell complex*. A 3-dimensional cell complex is formed by a finite set of k-dimensional cells (where $0 \leq k \leq 3$); a 0-cell is a vertex, a 1-cell an edge, a 2-cell a polygon, and a 3-cell a polyhedron. We name a $(k-1)$-cell incident to a k-cell a *facet* of it; a facet of a 3-cell (a room) is therefore a 2-cell that lies in its boundary (it is a wall or a door). A cell complex C has the following two conditions: (1) any facet of a k-cell in C is also in C; (2) the intersection of two cells c_i and c_j in C, denoted $c_i \cap c_j$, is either empty ($\emptyset$) or is a facet of both c_i and c_j.

A cell complex C can be represented by a graph $G = (V, E)$, where V is a set of vertices together with a set of edges E joining two distinct vertices. The embedding of this graph in the 3D Euclidean space ($\mathbb{R}^3$) creates a partitioning of $\mathbb{R}^3$ into cells; in IndoorGML this partitioning is called the *primal graph*.

The concept of *duality* (sometimes called Poincaré duality) is essential to IndoorGML. Duality can have many different meanings in mathematics, but it always refers to the translation or mapping in a one-to-one fashion of concepts or structures. In the context of a cell complex stored with a graph G, the *dual graph* of G, denoted G^*, is a mapping of the elements of G to other elements, and in IndoorGML G^* is the navigation graph.

If the embedding of G creates a cell complex C, then in theory the embedding of G^* can create a cell complex C^*. The mapping between C and C^* are the following: a k-cell c_i becomes a $(3-k)$-cell, which we denote c_i^*. For example: a 3-cell (a room) becomes a 0-cell (node) located somewhere inside the 3-cell; a 2-cell (a wall) becomes a 1-cell (an edge formed by the dual to the two rooms incident to the wall).

It should noticed that in IndoorGML we are only interested in a subset of C^*. This is because not all 2-cells in C have a dual 1-cell in C^*. For example, in Figure 2b, the facet between c_1 and c_2 (a wall) does not have an opening or door, and thus there is no edge linking c_1^* and c_2^* in the dual graph. Also, the 2-cells and 3-cells in C^* have no physical meaning: we simply use the dual graph for navigation, and to explicitly store the topological relationships between the 3-cells in C.

3 The methodology to validate IndoorGML files

The methodology I propose has six different steps (see Figure 3), and all of them should return "valid" for an IndoorGML model to be considered valid. The steps should be performed one after the other since invalid input could make the validation of another step fail (the software could crash).

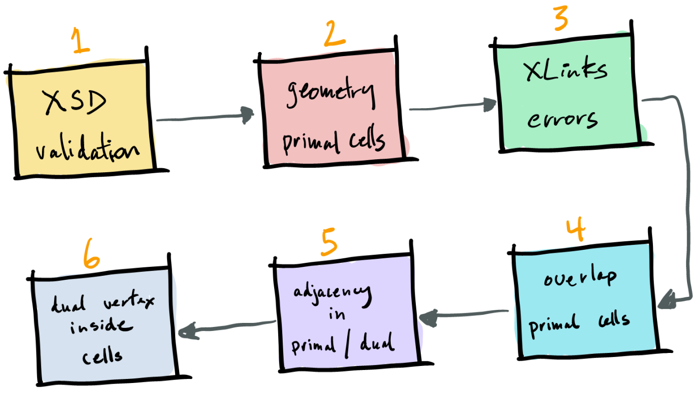


Figure 3: The six steps of the proposed methodology.da

I describe in this section the six steps separately, and for each I discuss related work. The first two steps reuse existing methodologies, while the other four are novel. Observe that the connectedness of the dual graph does not need to be validated, since this is not a requirements in IndoorGML. The local adjacencies between nodes in the graph are however validated, just not as a global test.

3.1 Schema validation

Most of the XML-based OGC standards, such as IndoorGML, have specifications in the form of text (usually a Word file) which is supplemented by XML schemas (`.xsd`). Those allow us to describe and encode a few constraints from the text, to verify that the syntax of an XML file is structured according to a few rules, for instance that certain elements are children of others, or that the coordinates of a solids have xyz values (and not only xy values). However, they can mislead users into thinking that the files are "perfect".

The schema validation is the first step in the validation because invalid files can often not be read by parsers. It was identified by the OGC CityGML Quality Interoperability Experiment (QIE) [OGC, 2016] as one of the four important aspect of validation; this project had exactly the same scope as the scope we are tackling in this paper, just for a different format.

For the purpose of this paper, we can consider the schema validation a "solved problem" since several tools exists.

3.2 Geometry of each 3-cell in the primal subdivision

IndoorGML uses the ISO19107 geometric primitives for representing the geometry of its objects [ISO, 2003], and a 3-cell in the primal has to be a `gml:Solid`; aggregates and composites types are not allowed. Observe that this means that inner boundaries (called cavities or voids) are allowed, and a 3-cell is not only a 2-manifold, which complicates its validation (see Ledoux [2013] for more details).

It should be noticed that while ISO19107 primitives do not need to be linear or planar, ie curves defined by mathematical functions are allowed, in practice IndoorGML does like CityGML and uses a subset of ISO19107 with the following two restrictions: (1) `GM_Curves` can only be *linear* (thus only `LineStrings` and `LinearRings` are used); (2) `GM_Surfaces` can only be *planar* (thus Polygons are used). This is not formally described in the standard, but there is a general understanding in the community about this [Li, 2020].

This step of the IndoorGML validation processes individually each 3-cell of the primal subdivision and verifies whether or not it is a valid gml:Solid. The methodology in Ledoux [2013] is reused as is, which means that the 3D primitives are validated hierarchically, starting with each ring forming a polygon. This allows us to report accurately an error to the user, and not just "this gml:Solid is invalid".

3.3 XLinks in primal/dual

The data model of IndoorGML relies heavily on XLinks (XML Linking Language), which are basically methods for creating internal (and external) links in an XML document. The primal and the dual subdivisions are linked with XLinks, and the dual graph is based on a complex set of XLinks. Because the XML schema validation cannot validate whether an XLink points to the XML element it should, special methods need to be defined.

IndoorGML stores the dual graph (navigation) in a non-standard data structure. Usually, for a graph, it would suffice to store all the nodes c_i^* in a matrix and for each edge between 2 nodes have an entry (with potentially attributes like distance, or the geometry of the edge if not a straight-line segment). However, in IndoorGML, each node c_i^* and edge f_i^* are created as separate elements, stored in separate lists. A node contains XLinks not to the adjacent nodes, but to the edges that are incident to it, and an edge contains an XLink to the 2 nodes it joins. There is therefore no direct access to the adjacent nodes of a given node, which is what navigation applications would expect. This implies that the possibility for having XLink errors/inconsistencies are high, because 2 separate data structures for the dual graph are maintained. Observe however that an IndoorGML dataset does not need to contain a dual graph and can solely contain a primal graph; in this case the duality XML elements are left empty.

Let c_a be a 3-cell in the primal graph, and c_a^* its dual node. We simply have to verify that the XLinks are *reciprocal*, that is that c_a points to c_a^*, and that c_a^* points to c_a (both XLinks are in the duality XML element of the corresponding elements).

Also, we have to verify the links between the nodes and edges in the dual graph. If c_a^* is incident to 3 edges (eg f_f^*, f_m^*, and f_p^*), then all 3 edges must also contain an XLink (in the connects XML element) to c_a^*. Observe that the edges are not directed, and thus they can be defined in any direction, and the search for c_a^* could be at the start or the end of a given edge.

All these rules are admittedly simple, but they need to be enforced and verified to ensure that an IndoorGML file is valid.

3.4 Overlap between 3-cells in the primal subdivision

The 3-cells in the primal graph should not overlap with each other. Two rooms never overlap, and so neither should their representations.

Specifically, two 3-cells c_a and c_b can have one facet f_m in common, but their interior must be disjoint ($c_a^o \cap c_b^o = \emptyset$). The validation is performed by finding all pairs of 3-cells in the primal (if there are n cells, then we need to test n^2 pairs) and simply perform the Boolean operation.

This validation step reuses and adapts the work in Ledoux [2018] for gml:CompositeSolid, and incorporates the concept of *overlap tolerance*. In practice, we can encounter solids that overlap by a very small amount, eg the overlapping volume could be 1cm^3 for a room and a door. While

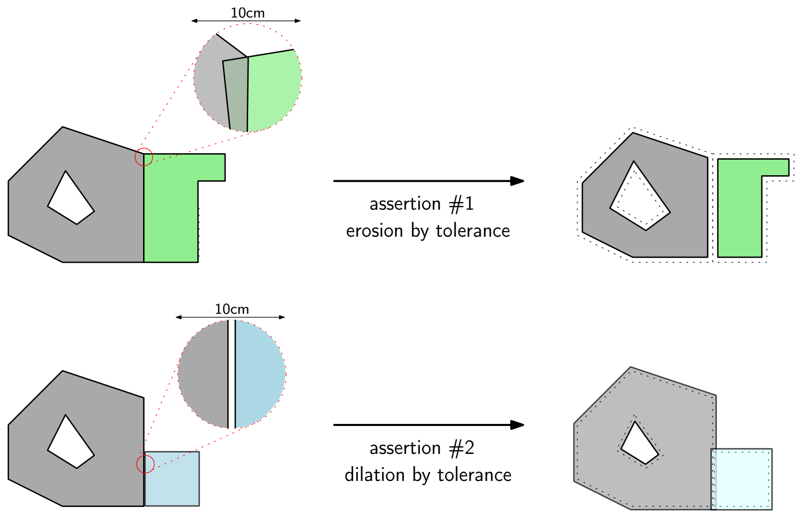


Figure 4: Example of how the tolerance is applied when verifying whether two `gml:Solids` overlap.

this is unwanted, using overlap tolerances allow us to indicate to the user where gross errors are. An overlap tolerance is a generalisation to 3D of the tolerance used for the 2D validation of polygon [van Oosterom et al., 2004]. As shown in Figure 4, the mathematical morphology theory in 3D [Serra, 1982] is used to erode and dilate `Solids` by a user-defined parameter. Erosion is performed when the overlap between `Solids` A and B is verified ($A^o \cap B^o = \emptyset$), and dilation when disjointness is verified ($A \cup B =$ one `gml:Solid`). These operations are realised by a series of operators that uses the Minkowski sum of a `gml:Solid` with a structuring element (a cube or dodecahedron in this case) [Boeters et al., 2015; Donkers et al., 2016].

3.5 Dual vertex inside the primal 3-cell

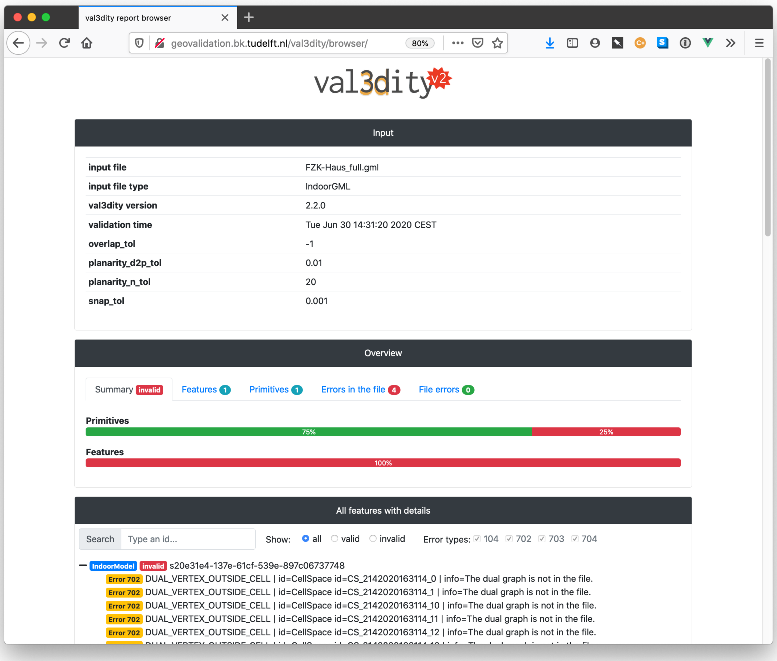
This is a simple test: for each c_a of the primal and its dual node c_a^* (if it exists) a point-in-polyhedron test is made. It should be located inside; notice that the location of the dual nodes in IndoorGML is usually simply the 'centre' of the solid (eg the centre of mass) since the exact location is not relevant. This means that extreme cases where the point is on the boundary should in practice not arise and the dual node should be unambiguously inside. This test is standard in 3D modelling.

3.6 Adjacency in primal = adjacency in dual?

This step validates the adjacency of cells in the primal and the dual. Basically, it verifies whether two adjacent 3-cells in the primal have their dual adjacent as well.

The validation is as follows. For each 3-cell c_a in the primal graph, we obtain its dual node c_a^* and then fetch its adjacent nodes in the dual graph, let them be c_p^* and c_q^* (2 in this case but there could be many). Then we have to verify that c_p and c_q (the dual cells) are adjacent to c_a. This is best performed by simply ensuring that the interior of the two 3-cells overlap, and the overlap tolerance defined in Section 3.4 can be used (the dilation part only).

As an example, consider Figure 2b, for the 3-cell c_3 we need to verify whether the dual to c_1^* and c_2^* are adjacent to c_3, but the adjacency between c_1 and c_2 would not be tested.



Input

input file	FZK-Haus_full.gml
input file type	IndoorGML
val3dity version	2.2.0
validation time	Tue Jun 30 14:31:20 2020 CEST
overlap_tol	-1
planarity_d2p_tol	0.01
planarity_n_tol	20
snap_tol	0.001

Overview

Summary invalid | Features 1 | Primitives 1 | Errors in the file 4 | File errors 0

Primitives 75% 25%

Features 100%

All features with details

Search: Type an id... Show: (● all) ○ valid ○ invalid Error types: ✓ 104 ✓ 702 ✓ 703 ✓ 704

```
— IndoorModel invalid s20e31e4-137e-61cf-539e-897c06737748
    Error 702 DUAL_VERTEX_OUTSIDE_CELL | id=CellSpace id=CS_2142020163114_0 | info=The dual graph is not in the file.
    Error 702 DUAL_VERTEX_OUTSIDE_CELL | id=CellSpace id=CS_2142020163114_1 | info=The dual graph is not in the file.
    Error 702 DUAL_VERTEX_OUTSIDE_CELL | id=CellSpace id=CS_2142020163114_10 | info=The dual graph is not in the file.
    Error 702 DUAL_VERTEX_OUTSIDE_CELL | id=CellSpace id=CS_2142020163114_11 | info=The dual graph is not in the file.
    Error 702 DUAL_VERTEX_OUTSIDE_CELL | id=CellSpace id=CS_2142020163114_12 | info=The dual graph is not in the file.
```

Figure 5: The web-based validation report browser.

4 Implementation and results

4.1 val3dity software now supports IndoorGML

The 6-step methodology described in Section 3 was implemented by extending the software *val3dity*, an open-source software to validate 3D primitives according to the international definitions of ISO19107. Its code is freely available under the GPLv3 license, and both binaries and a web-application are publicly available; see `https://github.com/tudelft3d/val3dity` for all the details.

The code is C++ (which ensures that it runs fast for large datasets and complex operations), and it is built upon solid and trusted libraries for manipulating spatial datasets: (1) the CGAL library[2], and (2) GEOS[3]. Because the geometric types and modules of CGAL do not follow the definitions of ISO19107, the geometric types available in different packages were modified and combined.

val3dity outputs a report that helps users identify and understand the errors. This report is in JSON, and can be browsed with the val3dity report browser in HTML (see Figure 5).

The errors are labelled and categorised, see Figure 6 for an overview; throughout the paper, Exxx refers to one specific error.

val3dity supports several generic formats (such as GML or OBJ), and also specific ones like CityJSON [Ledoux et al., 2019] and CityGML [OGC, 2012]. For those, specific validation rules have been developed, for instance those described in OGC [2016].

For IndoorGML, a new parser has been added to val3dity, and the validation methods have been implemented—all but the step 1 (Section 3.1) since it is assumed that the input file is

[2]`https://www.cgal.org/`
[3]`http://trac.osgeo.org/geos/`

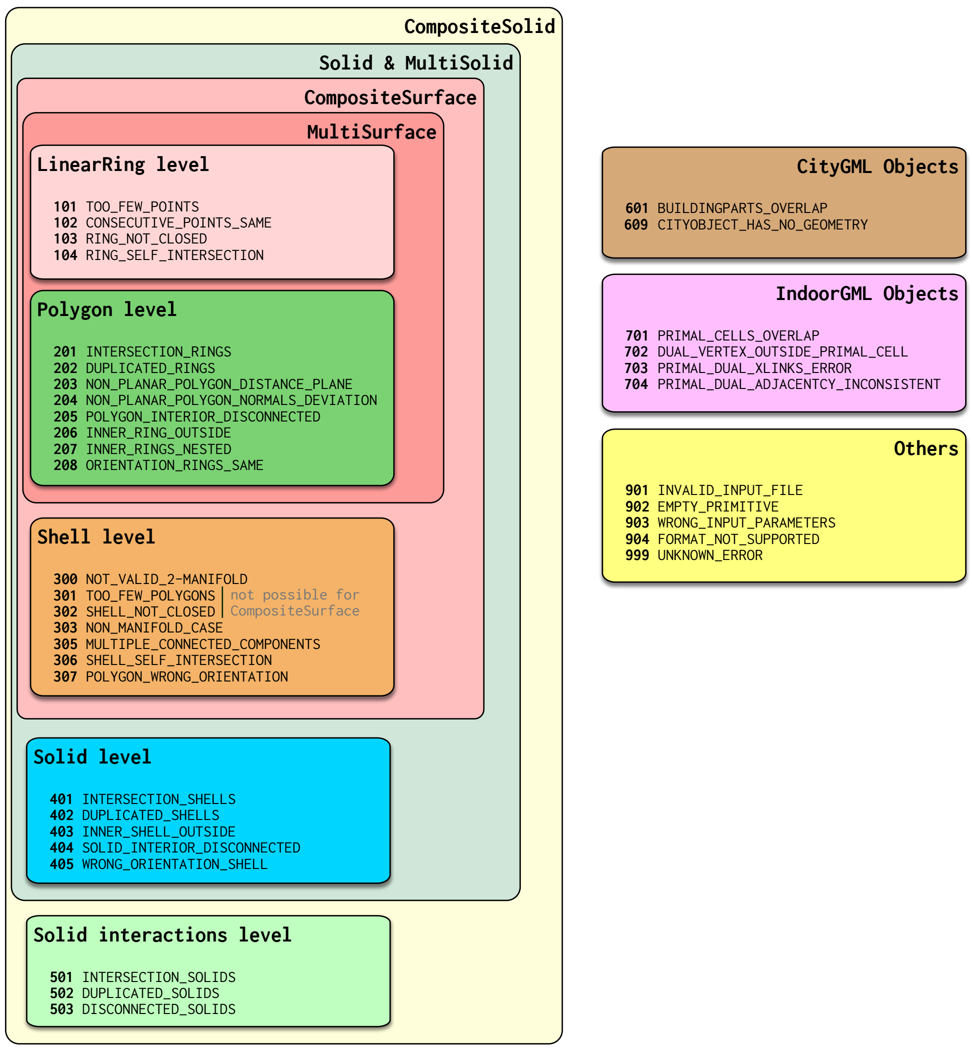


Figure 6: The 37 error codes of val3dity.

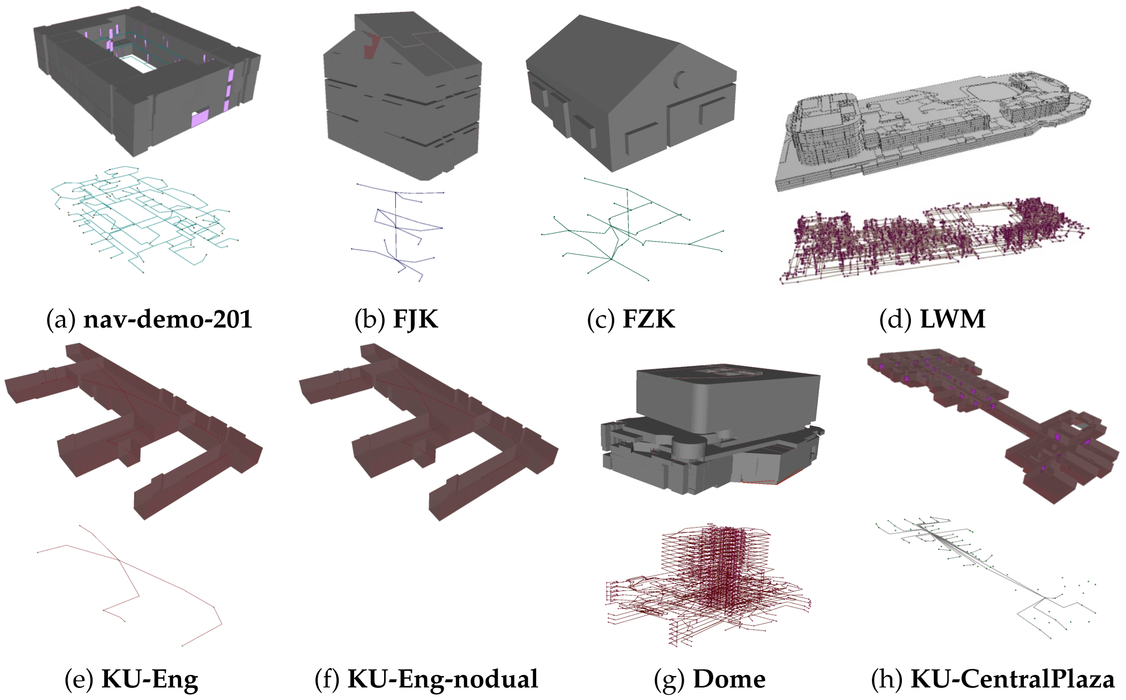


Figure 7: Datasets used for the experiments.

schema-valid. The different validation steps have been mapped either to existing error codes (eg for the geometries (E1xx–E4xx) and for invalid input files (E9xx)) or new error codes have been added (see E7xx). The E5xx are not possible for IndoorGML inputs because *gml:CompositeSolid*s are not allowed.

As is the case for other errors, when an error is reported, extra information is given to the practitioner so that they can fix the problem. An example is if E702 (`DUAL_VERTEX_OUTSIDE_PRIMAL_CELL`) is reported, then the identifier of the 3-cell (its `gml:id`) would also be reported.

4.2 Experiments with real-world datasets

All the IndoorGML datasets that I could find were validated. I asked on the OGC IndoorGML-SWG mailing, and I downloaded the four on the official IndoorGML webpage[4].

The resulting eight datasets are shown in Figure 7, and their details are available in Table 1. Observe that there are more existing IndoorGML datasets, eg those created in the context of the OGC IndoorPilot [OGC, 2019], but because of licenses and copyright they could not be used for this article. Also, several files of the same buildings, but with different dates, exist and I kept the most recent version for the tests.

The datasets (the original IndoorGML files) and their validation reports are available at `https://github.com/hugoledoux/indoorgml_validation` (public repository); notice that one dataset (*Dome*) cannot be freely distributed and is thus not in the repository.

Each of the dataset was validated with *val3dity* v2.2.0[5]. The default parameters were used[6], but these can be modified.

[4]`http://indoorgml.net/resources/`

[5]all versions available at `https://github.com/tudelft3d/val3dity/releases`

[6]for snapping vertices (`--snap_tol=0.001`) and for detecting overlap (`--overlap_tol=0.0`), see `https://val3dity.readthedocs.io/en/latest/usage/` for their exact meaning

Table 1: Datasets used for the experiments (see Figure 7).

	cells	nodes	edges	size	extension	open data
nav-demo-201 (1)	80	80	184	1.7MB	Navigation	✓
FJK (1)	14	14	30	344KB	none	✓
FZK (2)	24	24	24	137KB	none	✓
LWM (1)	3496	4893	5360	47MB	none	✓
KU-Eng (3)	7	7	6	52KB	none	✓
KU-Eng-nodual (3)	7	0	0	111KB	none	✓
Dome (3)	2115	2109	5026	26MB	none	×
KU-CentralPlaza (3)	76	88	108	803KB	PSExt+NonNav	✓

(1) available at `http://indoorgml.net/resources/`
(2) obtained from Diakité [2020]
(3) obtained from Li [2020]

Table 2: Validation overview.

		primal cells		IndoorGML errors				
	schema	errors	%valid	E701	E702	E703	E704	time
nav-demo-201	✓	E104/E401	95%	6	0	2	0	6.2s
FJK	✓	E103	14%	0	0	0	0	0.2s
FZK	✓	E104	75%	0	24	24	24	0.2s
LWM	✓	E102/E104	99%	398	0	3491	0	402.6s
KU-Eng	×	—	—	—	—	—	—	[crash]
KU-Eng-nodual	✓	E302/E405	0%	0	0	0	7	0.0s
Dome	✓	—	100%	1206	0	0	556	443.5s
KU-CentralPlaza	✓	E405	32%	1	0	0	0	0.2s

Table 2 shows the overview results for the eight datasets. The validations were performed on a standard laptop (MacBook Pro, 2.3GHz 8-core, 32GB memory), and the time shown is for reading the input file, validating it, and writing to disk the overview report.

General comments. None of the datasets are valid. This is not surprising because, notoriously, 3D datasets have several errors: the software to create and eventually fix/repair datasets in 3D are not at the same level as those in 2D, and topological data structures for 3D objects (which means non-manifold modelling must be used) are very rare. It is important to notice that the errors shown are not all the errors for a given datasets. As explained in Section 3.2, the validation is hierarchical to avoid "cascading errors". Concretely, this implies that if a given cell c contains an error (say E302) then the other validation tests that would require the volume of c are not performed (eg no point-in-polyhedron tests are performed, nor is the adjacency between 2 cells). This means that if the geometry of c was fixed, then potentially (new) errors E7xx could be reported. The time taken to validate the datasets should also consider that "the better the quality of a dataset the longer it will take to validate"; if a dataset has none of its cells valid, then many of the complex validation tests for IndoorGML will be skipped.

Schema invalid. All the files are schema-valid, except one (*KU-Eng*). While val3dity can in theory recover from small schema errors in the file, the error in *KU-Eng* meant that the parsing of the file was wrong, and the process crashed. The error is actually the simplest possible

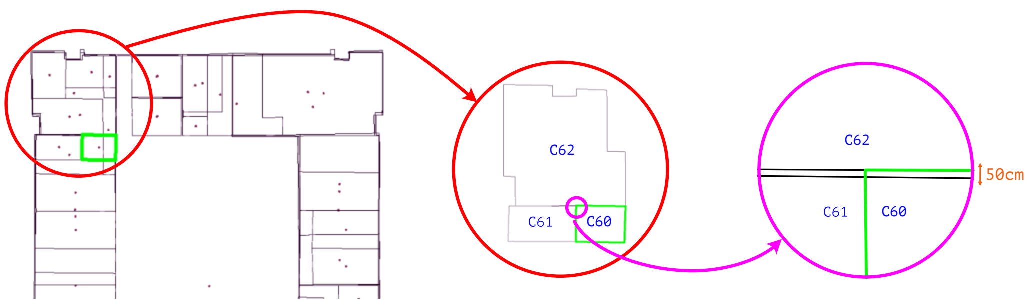


Figure 8: Some cells of *nav-demo-201* projected to the 2D plan; the identifiers shown are `gml:id` of the cells.

almost: a wrong capitalisation of an XML element: `core:spaceLayer` is used instead of the correct `core:SpaceLayer` (notice the capital 's'; XML is case-sensitive). One might argue that the parser could and should recover from such a pointless error, however observe that the IndoorGML element `core:spaceLayer` is valid and must be used at for other elements.

Cells contain many (simple) geometric errors. Many datasets contain simple geometric errors (E100–E500) that could have been easily avoided, eg E102 and E103. The E405, affecting 32% of the cells in *KU-CentralPlaza*, could also have been fixed easily: it suffices to inverse the orientation of the surfaces of each cell. The E302 means that one surface is missing from a solid, and thus is more complex to fix in practice. While often ignored and/or forgotten [Ledoux, 2013], the solid representing a cell can contain interior boundaries (cavities). The dataset *nav-demo-201* contains 3 cells with such solids, and one of them is invalid (E401).

Large datasets. For large datasets (having several cells), since the number of tests to be performed is quadratic (E701 requires testing each cell against all the other cells), the number of intersections tests to be performed can become very large. For *LWM*, which contains 3496 cells, the number of intersection tests is thus 12.2 millions. val3dity has therefore been modified so that above a certain number of cells, the cells are spatially indexed with an *AABB tree*[7], each element of the tree is the axis-aligned bounding box of the cell. Before the intersection test is attempted (a costly operation in 3D), we simply verify whether the 2 bounding boxes overlap, if not then the 2 cells can not have their interior overlap. This speeds up greatly the processing, but it is in practice still rather slow for the very large datasets; *LWM* takes for instance a bit more than 8 minutes to be processed.

Overlap between cells (E701). The E701 is present in 4/8 datasets. In *nav-demo-201* there are 6 pairs of cells overlapping; Figure 8 shows one example where the cells are overlapping by several centimetres (about 18cm in reality). Notice however that the default validation values used assume that the overlap tolerance is 0cm, and in the datasets there are small gaps between cells on different floors. If the overlap tolerance is set to 5cm, then only the overlaps shown in Figure 8 remain (this means that there is $< 5cm$ between several cells). This however slows down the process since the computations required are complex, for *nav-demo-201* the running time goes from 6.2s to 49s (almost 8X slower). The dataset *LWM* has 398 pairs of cells that are overlapping, and most of them are blunders where smaller cells are contained within larger cells, Figure 9 shows how 4 cells interact. The dataset *Dome* contains also a rather large number of E701.

[7]`https://doc.cgal.org/latest/AABB_tree/index.html`

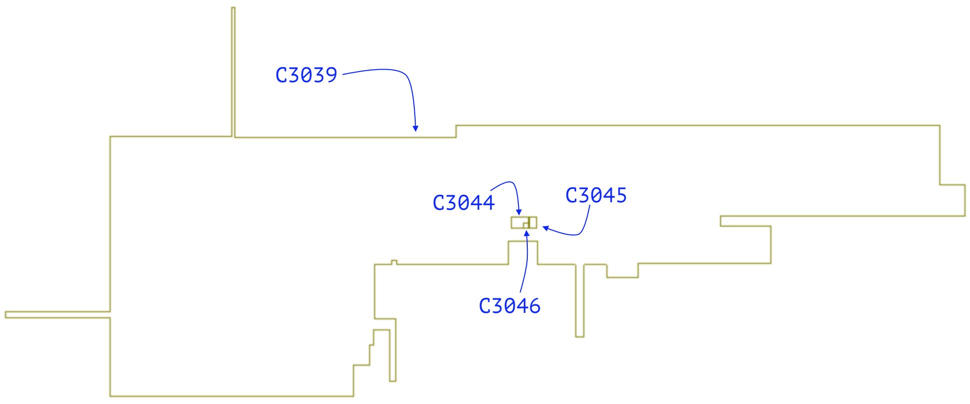


Figure 9: Top-view of 4 cells in *LWM* on the same floor: cell `C3039` overlap with `C3044`, `C3045`, and `C3046`. There are 398 pairs of cells overlapping for this dataset.

Errors in primal/dual XLinks (E703). The dataset *LWM* contains a very high number of E703. This is caused by the fact that the XLinks from the primal cells to the dual vertices are very often wrong, ie the `gml:id` of the vertex does not exist in the file. Alarmingly, none of the vertices in that datasets have a `duality` element, which means that primal cells refer to a dual node, but the nodes do not refer back to the cell. This means that the E704 cannot be tested, and this results in 0 errors. IndoorGML specifications do not enforce this for vertices, so an error cannot be reported. Also, there are other cases where a node reference an edge (Transition) that is non-existent, eg *nav-demo-201* contains two such cases.

Tolerances and error E704. The overlap tolerance also has an influence on other errors, for example E704. For *nav-demo-201*, if `--overlap_tol=0.01` is used, instead of 25 E704 we obtain only but 1. As pointed out earlier, the running time of the validation process is 8X longer.

5 Discussion

The six validation tests presented in this paper, and implemented in the software val3dity, allow us to verify whether the information contained in an IndoorGML file is structured and compliant with the official specifications [OGC, 2014].

As shown in the paper, none of the existing IndoorGML files are valid, at least those that are publicly available or those that could be legally used for the experiments. "Is that a problem in practice?" one might ask. The answer to this question is a complex one, and is very much the same as that for other types of 3D geoinformation datasets like city models: it depends on the application [Biljecki et al., 2015, 2016].

City models, like CityJSON [Ledoux et al., 2019] and CityGML [OGC, 2016], are often used for visualisation and visibility-based applications, and for those most of the validity and quality criteria are not very important, ie if the surfaces are valid then most likely the application will not suffer.

However, IndoorGML was not developed as a visualisation format, but (mostly) as one to support indoor navigation. The navigation graph should therefore be valid, but as Section 4 shows, this is often not the case with tested files.

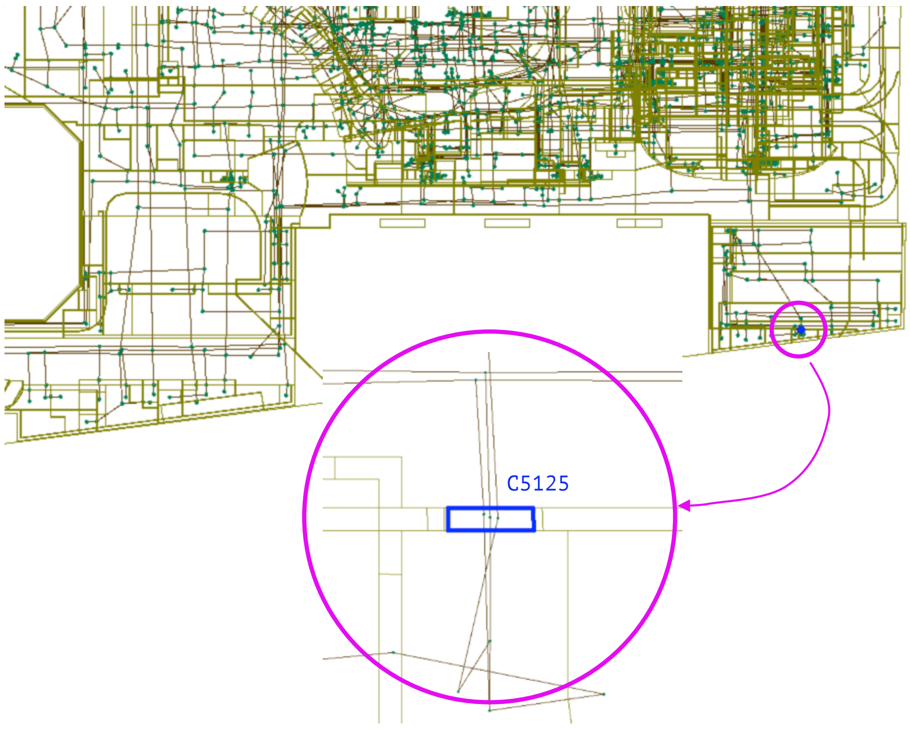


Figure 10: One of the many examples where dataset *LWM* has issues with the dual nodes. Cell "C5125" has 3 nodes inside, but none of them are linked by the cell (which has as a dual a non-existent node).

Most of the complex validation errors reported are due to the topological relationships between the cells (E701 and E704). While IndoorGML does not mandate that adjacent cells be perfectly adjacent (just that they should not overlap), in all the sketches (eg see Figure 1) and examples one can *see* that they are adjacent. Or are they? According to the experiments made in this paper, there can be gaps that can be at the sub-centimetre level. Using the overlap tolerance (Section 3.4, and option `overlap_tol` in val3dity) solves this issue. This however slows down tremendously the running-time, and setting the value for the tolerance can be tricky. I believe a more efficient implementation would be possible, but waiting 30s or more for a small IndoorGML files containing 80 cells (such as *nav-demo-201*) can be a drawback.

Furthermore, I have identified a few particularities in the IndoorGML files, which, while were not reported as errors (because the OGC specifications v1.0 do not mention those cases), should be highlighted as potential issues. Those could cause downstream applications to not be able to function properly. One of them is: should there be one and only one dual node for a given cell? The XML element for a cell can only store one link to a node (element `duality`), but what if the dual graph contains more nodes, and only a subset of those have a link back to a primal cell? The dataset *LWM* contains several such cases. It is very easy to see by looking at the statistics of the dataset in Table 1: there are 3496 cells, but 4893 nodes. Figure 10 shows the issue for one specific cell, among many others. The dataset *LWM* also contains three different dual graphs (this is possible in the specification, and it is the only file in the experiments having more than one graph), but only two of the three have edges and are linked the primal. Indeed, there is a graph composed solely of nodes, and those nodes represent the locations of fire extinguishers in the building (the Public Safety Extension is used). I would argue that storing those where the developers expect nodes of the dual is a poor design choice in IndoorGML. Similarly to this, the dataset *KU-Eng* has no dual at all: should that be an error? At this moment, since the specifications do not enforce the dual, no errors were returned, but perhaps warnings should be used in the future.

Finally, as future work, I plan to fix the bugs in val3dity that will be raised by the influx of new IndoorGML files that will surely be produced by the community in the near future.

Acknowledgements

This research was supported by a grant (20NSIP-B135746-04) from National Spatial Information Research Program (NSIP) funded by Ministry of Land, Infrastructure and Transport of Korean government, and it has received funding from the European Research Council (ERC) under the European Unions Horizon2020 Research & Innovation Programme (grant agreement no. 677312 UMnD: Urban modelling in higher dimensions).

References

Alam N, Wagner D, Wewetzer M, von Falkenhausen J, Coors V, and Pries M (2014). Towards automatic validation and healing of CityGML models for geometric and semantic consistency. In *Innovations in 3D Geo-Information Sciences*, pages 77–91. Springer International Publishing.

Alattas A, Zlatanova S, Van Oosterom P, and Li KJ (2018). Improved and more complete conceptual model for the revision of IndoorGML. In Winter S, Griffin A, and Sester M, editors, *10th International Conference on Geographic Information Science (GIScience 2018)*, volume 114 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:12. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany.

Biljecki F, Ledoux H, Du X, Stoter J, Soon KH, and Khoo VHS (2016). The most common geometric and semantic errors in CityGML datasets. volume IV-2/W1 of *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 13–22. Athens, Greece.

Biljecki F, Stoter J, Ledoux H, Zlatanova S, and Çöltekin A (2015). Applications of 3D city models: State of the art review. *ISPRS International Journal of Geo-Information*, 4(4):2220–9964. doi:http://dx.doi.org/10.3390/ijgi4042842.

Boeters R, Arroyo Ohori K, Biljecki F, and Zlatanova S (2015). Automatically enhancing CityGML LOD2 models with a corresponding indoor geometry. *International Journal of Geographical Information Science*, 29(12):2248–2268.

Colley P, Kazar BM, Kothuri R, van Oosterom P, and Ravada S (2017). Validation of three-dimensional geometries. In *Encyclopedia of GIS*, pages 2398–2405. Springer International Publishing.

Coors V, Betz M, and Duminil E (2020). A concept of quality management of 3D city models supporting application-specific requirements. *Journal of Photogrammetry, Remote Sensing and Geoinformation Science (PFG)*, 88:3–14. doi:10.1007/s41064-020-00094-0.

Dao THD and Thill JC (2017). Three-dimensional indoor network accessibility auditing for floor plan design. *Transactions in GIS*, 22(1):288–310.

Davis M (2003). Java conflation suite. Technical report, Vivid Solutions. Draft available at `http://www.vividsolutions.com/jcs/`, has never been completed.

Diakité AA and Zlatanova S (2017). Spatial subdivision of complex indoor environments for 3D indoor navigation. *International Journal of Geographical Information Science*, 32(2):213–235.

Diakité A (2020). Personal communication.

Donkers S, Ledoux H, Zhao J, and Stoter J (2016). Automatic conversion of IFC datasets to geometrically and semantically correct CityGML LOD3 buildings. *Transactions in GIS*, 20(4):547–569.

Goetz M and Zipf A (2011). Formal definition of a user-adaptive and length-optimal routing graph for complex indoor environments. *Geo-spatial Information Science*, 14(2):119–128.

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

Hashemi M (2018). Emergency evacuation of people with disabilities: A survey of drills, simulations, and accessibility. *Cogent Engineering*, 5(1).

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

Kang HK and Li KJ (2017). A standard indoor spatial data model—OGC IndoorGML and implementation approaches. *ISPRS International Journal of Geo-Information*, 6(4):116.

Ledoux H (2013). On the validation of solids represented with the international standards for geographic information. *Computer-Aided Civil and Infrastructure Engineering*, 28(9):693–706. ISSN 1467-8667. doi:http://dx.doi.org/10.1111/mice.12043.

Ledoux H (2018). val3dity: validation of 3D GIS primitives according to the international standards. *Open Geospatial Data, Software and Standards*, 3(1):1. doi:http://dx.doi.org/10.1186/s40965-018-0043-x.

Ledoux H, Ohori KA, Kumar K, Dukai B, Labetski A, and Vitalis S (2019). CityJSON: a compact and easy-to-use encoding of the CityGML data model. *Open Geospatial Data, Software and Standards*, 4(4). doi:http://dx.doi.org/10.1186/s40965-019-0064-0.

Li KJ (2020). Personal communication.

Mulder D (2015). *Automatic repair of geometrically invalid 3D city building models using a voxel-based repair method*. Master's thesis, 3D geoinformation group, Delft University of Technology, Delft, the Netherlands.

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

OGC (2014). OGC IndoorGML. Open Geospatial Consortium inc. Document 14-005r3, version 1.0.

OGC (2016). OGC CityGML quality interoperability experiment. Open Geospatial Consortium inc. Document OGC 16-064r1.

OGC (2019). OGC Indoor mapping and navigation pilot engineering report. Technical report, Open Geospatial Consortium. Document OGC 16-064r1.

Park S, Kim S, and Yu K (2018). Designing of indoor linkable pedestrian network data model for the transportation vulnerable. In *Proceedings 2nd International Conference on Digital Signal Processing (ICDSP)*. ACM Press.

Pédrinis F, Morel M, and Gesquière G (2015). Change detection of cities. In *3D Geoinformation Science*, pages 123–139. Springer International Publishing.

Serra JP (1982). *Image Analysis and Mathematical Morphology*. Academic Press.

Steuer H, Machl T, Sindram M, Liebel L, and Kolbe TH (2015). *Voluminator—approximating the volume of 3D buildings to overcome topological errors*, pages 343–362. Lecture Notes in Geoinformation and Cartography. Springer Science.

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.

Wagner D, Alam N, Wewetzer M, Pries M, and Coors V (2015). Methods for geometric data validation of 3D city models. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, XL-1-W5:729–735.