

Article

Incorporating topological representation in 3D City Models

Stelios Vitalis ^{*,‡} Ken Arroyo Oho [‡] and Jantien Stoter [‡]

3D geoinformation group, Delft University of Technology; s.vitalis@tudelft.nl; g.a.k.arroyoohori@tudelft.nl; j.e.stoter@tudelft.nl

* Correspondence: s.vitalis@tudelft.nl

‡ All authors contributed equally to this work.

Version May 5, 2019 submitted to Preprints

Abstract: 3D city models are being extensively used in applications such as evacuation scenarios and energy consumption estimation. The main standard for 3D city models is the CityGML data model which can be encoded through the CityJSON data format. CityGML and CityJSON use polygonal modelling in order to represent geometries. True topological data structures have proven to be more computationally efficient for geometric analysis compared to polygonal modelling. In a previous study, we have introduced a method to topologically reconstruct CityGML models while maintaining the semantic information of the dataset, based solely on the combinatorial map (C-Map) data structure. As a result of the limitations of C-Map's semantic representation mechanism, the resulting datasets could suffer either from semantic information loss or the redundant repetition of them. In this article, we propose a solution for a more efficient representation of both geometry, topology and semantics by incorporating the C-Map data structure in the CityGML data model and implementing a CityJSON extension to encode the C-Map data. In addition, we provide an algorithm for the topological reconstruction of CityJSON datasets to append them according to this extension. Finally, we apply our methodology to three open datasets in order to validate our approach when applied to real-world data. Our results show that the proposed CityJSON extension can represent all geometric information of a city model in a lossless way, providing additional topological information for the objects of the model.

Keywords: 3D city model; Topology; Combinatorial Map; Linear Cell Complex; CityJSON; CityGML

1. Introduction

3D city models have been increasingly adopted in modern analysis of urban spaces, such as the simulation of evacuation scenarios [1] and optimisation of energy consumption for city districts [2,3]. Their key benefit is that they can describe complex 3D geometries of city objects, such as buildings, vegetation and roads; and their semantic information, such as their purpose of use and year of construction.

CityGML is the most commonly used data model for the representation of 3D city models [4], which can be encoded in JSON through the CityJSON data format. The data model incorporates the Simple Feature Specification (SFS), which describes the geometric shapes by their boundaries through a method that is referred as "polygonal modelling". While polygonal modelling is generally considered a robust representation of 2D data, it has been proven inefficient when representing 3D objects. This reflects to the limited number of 3D processing algorithms that can be easily applied to polygonal modelling [5].

Topological data structures have been introduced in GIS as an alternative to polygonal modelling. Their main characteristic is that they explicitly describe the adjacency and incidence relationships between geometric objects. Those relationships can improve the performance of geometric processing. For example, Maria *et al.* [6] exploited the topological properties of geometries in order to improve the efficiency of ray tracing in architectural models. Furthermore, topological data structures have the

37 ability to scale to higher dimensions without adding unnecessary complexity [7]. Therefore, typical
38 GIS operations can be described as dimension-agnostic algorithms which can be applied in an arbitrary
39 number of dimensions. For example, Arroyo Ohori *et al.* [8] have proposed a solution for the extrusion
40 of objects of any number of dimension.

41 For this reason, we have investigated the use of ordered topological structures and, more
42 specifically, combinatorial maps (C-Maps) as an alternative to the SFS for the representation of
43 geometric information in 3D city models. C-Maps combine the powerful algebra of geometric simplicial
44 complexes with the ease of construction of polygonal modelling [7]. Regarding the practical aspect,
45 they are implemented in a software package as part of CGAL¹ and they are efficient with respect to
46 memory usage [9]. While C-Maps originally store only topological relationships between objects, they
47 can be easily enhanced with the association of coordinates to vertices, which results in a linear cell
48 complex (LCC) that incorporate both geometric and topological information.

49 LCCs based on C-Maps have been used before in 3D city models. Diakit  *et al.* [10] have proposed
50 a methodology on the topological reconstruction of existing buildings that are represented through
51 polygonal modelling. They, then, use the topological information in order to simplify the building's
52 geometry. This approach is based on the extraction of a soup of triangles from the original geometry
53 which are later stitched together according to their common edges in order to identify the topological
54 relationships. During this intermediate step, the semantic information of the original model, such as
55 hierarchical relationships, are lost as the soup does not retain the information of the original model.
56 Diakit  *et al.* [11] have further refined this process and applied it to BIM and GIS models, in order to
57 exploit the topological information to identify specific features of buildings. Although this application
58 includes the reconstruction of CityGML models, it results in a semantic-free model where the original
59 city objects' subdivision is lost.

60 Previously, we have introduced a methodology for the topological reconstruction of CityGML
61 models to LCCs based on C-Maps with preservation of semantics [12]. Due to the fact that this
62 methodology was relying solely on the C-Maps data structure for the representation of all information
63 of the 3D city model, the resulting model would suffer from either occasional loss of the semantic
64 subdivision of city object, or a redundancy of information. For example, in that article we have
65 topologically reconstructed the 3D city model of *Agnesebuurt*, a neighbourhood of Rotterdam, which
66 was missing intermediate walls between adjacent building. As a consequence of the missing walls,
67 multiple individual buildings were merged under the same volumes in the resulting C-Map. This
68 causes the loss of semantic information of some buildings during the reconstruction as only one city
69 object's information could be attached in the resulting volume.

70 In this paper we propose an improved methodology for the topological representation of CityGML
71 models, in order to avoid the limitations of semantics representation in the C-Maps data structure
72 and the limitations of topological representations in CityGML. In order to achieve that, we integrate
73 the original CityGML data model with C-Maps in order to combine the semantic-representation
74 capabilities of the first with the benefits of a topological data structure. This topologically-enhanced
75 data model is implemented in CityJSON through an extension. We, also, develop an algorithm in order
76 to transform existing CityJSON datasets. Finally, we apply our algorithm to several open datasets in
77 order to assess the robustness of our method and evaluate the ability of the proposed data model to
78 represent the peculiarities of various datasets.

¹ The Computational Geometry Algorithms Library (<http://www.cgal.org>)

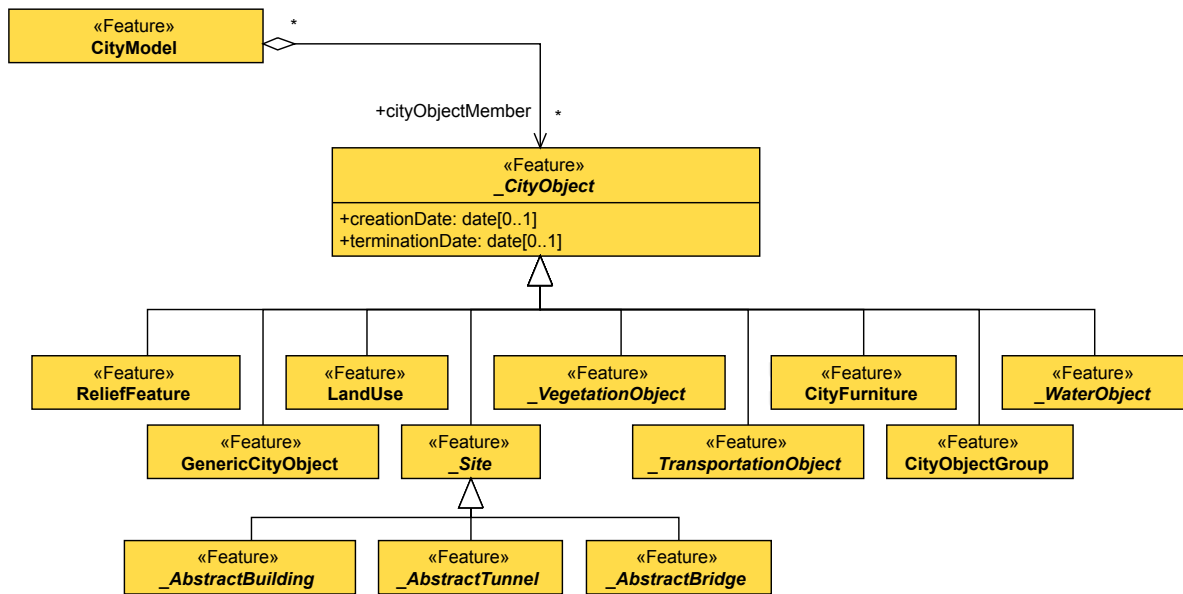


Figure 1. The UML diagram that describes CityGML's top level class hierarchy of city objects.

79 2. Related work and background information

80 2.1. The CityGML data model

81 CityGML is a data model and an XML-based format that has been standardised through
 82 the Open Geospatial Consortium (OGC) in order to store and exchange 3D city models [4]. It
 83 defines an object-oriented approach for the representation of city objects, utilising techniques such as
 84 polymorphism in order to provide enough flexibility.

85 In CityGML, a city model contains a number of city objects of different types, all of which inherit
 86 from the basic abstract class *CityObject*. Different types of objects can be represented through derived
 87 classes which can be: (a) composite objects, such as *CityObject Group*; (b) specialised abstract classes,
 88 such as *AbstractBuilding*; or (c) actual city objects, such as *CityFurniture* and *LandUse* (Figure 1). Given
 89 that a CityGML dataset has a tree structure, the objects can be either listed as immediate child nodes in
 90 the model or they can be represented in a deeper layout by grouping objects using the *CityObjectGroup*
 91 class.

92 CityGML is a schema that extends the geographic markup language (GML) [4], therefore it follows
 93 GML's geometric representation which is based on the simple feature specification (SFS) [13]. Every
 94 *CityObject* contains one or more *Geometry* objects according to the SFS representation. They *Geometry*
 95 object can be extended through composition, therefore a geometry can be a primitive or a composite
 96 object of multiple geometries.

97 According to the CityGML data model, city object semantics are represented through two
 98 mechanisms: object types and attributes. The type of a *CityObject* is derived from the class used in order
 99 to represent it. Then, additional information for every *CityObject* can be stored in the model through
 100 attributes which are described by the CityGML specification. A user can enhance the data model with
 101 additional attributes related to their domain requirements, either by using the *GenericAttribute* class or
 102 by developing an application domain extension (ADE).

103 2.2. The CityJSON data format

104 CityJSON is a data format which uses the JavaScript Object Notation (JSON) encoding in order to
 105 implement a subset of the CityGML data model. Its goal is to serve as an alternative to the CityGML

106 data format, which is verbose and complex due to its GML (and, thus, XML) nature. For example,
107 there are at least 26 different ways to encode a simple four-points' square in GML².

108 CityJSON uses a simpler structure that allows for less ambiguity and verbosity. First of all, it
109 uses the JSON encoding, which is easier to parse and write. This is due to its representation, which can
110 be mapped directly to the data structures that are supported by most modern programming languages:
111 key-value pairs (known as maps or dictionaries) and arrays. Second, CityJSON promotes a "flat" list
112 of city objects, while hierarchy can be implied through internal attributes (through the parents and
113 children attributes).

114 Similar to ADEs for the CityGML data format, CityJSON also provides an extension mechanism
115 for defining domain-specific city objects and attributes. Through CityJSON Extensions, one can
116 introduce new type of city objects or append existing ones with attributes related to the subject of the
117 extension.

118 2.3. Combinatorial Maps

119 A combinatorial map (C-Map) is a data structure that represents a topological partition of
120 n -dimensional space[14]. The partitions of this space are called *cells* and they are of any dimension in
121 this space. For example, in a 3-dimensional space a C-Map denotes 0-cells which are vertexes, 1-cells
122 which are edges, 2-cells which are facets and 3-cells which are volumes.

123 C-Map represent the space through *dart* elements. Darts are similar to half-edges for an arbitrary
124 amount of dimensions: every part of an edge that belongs to every possible combination of i -cells
125 ($0 < i \leq n$) is a dart. Darts are connected through β_i links (where $0 \leq i \leq n$) so that every dart contains
126 one $\beta_i \forall i \in \{1, \dots, n\}$. A β_i is a link to the next dart in the i -cell. For example, in a 4D C-Map a β_3
127 of the dart d links to the dart that belongs to the same edge (1-cell) of the same facet (2-cell) of the
128 same polychoron (4-cell) as d_1 , but is part of the adjacent volume (3-cell). A *null* dart (denoted as \emptyset) is
129 introduced to the C-Map in order to represent darts that do not have adjacent cells. A dart with no
130 adjacent i -cell has $\beta_i = \emptyset$ and is called i -free.

131 In order to modify C-Maps we define the *sewing* operation, according to which pairs of
132 corresponding darts of two i -cells are linked in one dimension. A i -sew operation associates together
133 two i -cells along their common incident $(i - 1)$ -cell. This means that the β_i 's of every pair of darts
134 along the two $(i - 1)$ -cells has to be linked.

135 Cells in a C-Map can be associated to information through a mechanism of attributes. A dart
136 holds a set of attributes, one for every dimension of the C-Map which is called the i -attribute of the
137 dart. For example, in order to set a property of a facet (e.g. colour), one can set this colour value to the
138 2-attribute of every dart of this facet (2-cell).

139 By associating the vertexes of the C-Map with point of a n -dimensional geometric space and
140 assuming that all geometries of the C-Map are linear we can represent a linear cell complex (LCC).
141 Then, this LCC contains both geometric and topological information for the space.

142 3. Topological representation of 3D city models

143 In order to represent the topological information of city objects in a CityJSON file we followed
144 two steps: first, we introduced the LCC entities into the CityGML data model; and second, we
145 developed a CityJSON extension that provides the necessary encoding instruction in order to store
146 those information in a CityJSON file. We have developed the extension definition according to the
147 respective CityJSON specification³. The CityJSON extension definition is available as open source in
148 GitHub⁴.

² <https://erouault.blogspot.com/2014/04/gml-madness.html>

³ <http://www.cityjson.org/en/0.9/extensions/>

⁴ <https://github.com/tudelft3d/cityjson-lcc-extension>

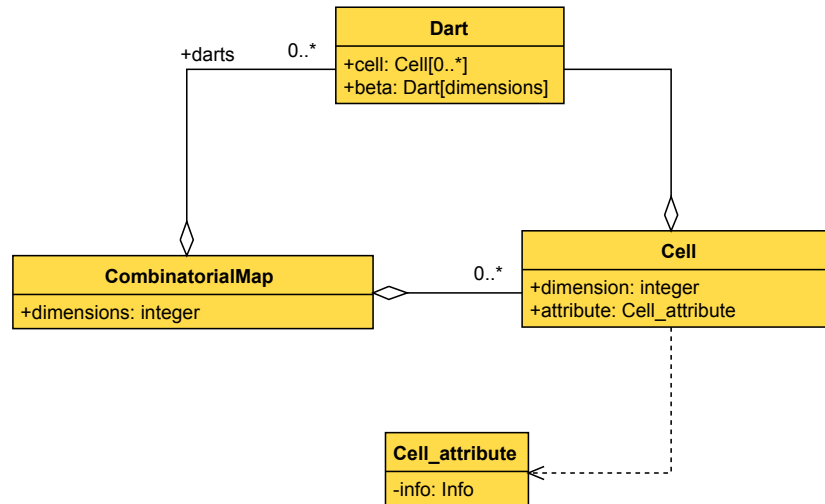


Figure 2. The UML diagram that describes the Linear Cell Complex implementation of CGAL

149 3.1. Data model

150 First, it must be possible to store darts in the city model. Second, every dart has to be linked
 151 with: (a) the point that defines the location of the dart's 0-cell, (b) the city object that is associated to
 152 the dart's 2-cell, (c) the semantic information associated with the dart's 2-cell, and (d) its β_i darts. It
 153 must be noted that β_0 is not essential for the storage of the map, as it can be implied by the β_1 's of the
 154 structure. Therefore, for a three-dimensional LCC β_1 , β_2 , and β_3 are only required to be stored.

155 We decided to associate surfaces (2-cells) with the city objects' semantic information, which might
 156 seem a counter-intuitive solution. Initially, volumes (3-cells) might seem as a more suitable match for
 157 association with city objects. After all, a city object is normally composed of volumes. Unfortunately,
 158 as we proved in Vitalis *et al.* [12], that is not always the case. In some city models there can be multiple
 159 city objects that topologically belong to one volume.

160 While in Vitalis *et al.* [12] we have proposed a way of forcing 3-cells to be divided into multiple
 161 volumes, when their 2-cells belong to different city objects, the final result is not correct from a
 162 topological perspective. The key benefit of using a LCC data structure is to store the topological
 163 relationships of a city model's geometry. Consequently, such a solution would largely undermine
 164 the benefits of using a topological representation in the first place. Therefore, we have decided to
 165 associate city objects' semantic information with 2-cells in order to be able to maintain the binding of
 166 information in cases where one volume is associated with multiple city objects. This way, we ensure
 167 topological consistency and retain a complete association of semantics with the LCC.

168 We have designed a data model that represents the LCC information through the Dart class
 169 (Figure 3). The class contains the necessary information as attributes: (a) the `vertexPoint` attribute
 170 points to the `Vertex` object that stores the coordinates of the 0-cell; (b) the `parentCityObject` attribute
 171 points to the `CityObject` associated with the dart's 2-cell; (c) the `semanticSurface` attribute links
 172 the dart's 2-cell with the semantic information of the incident surface; and (d) the `beta` attribute is a
 173 three-element array of `Dart` objects, representing β_1 , β_2 and β_3 .

174 While the Dart class and its attributes can represent a complete linear cell complex, they only
 175 preserve one-way links from the C-Map to the city model. Nevertheless, it is equally important to
 176 be able to identify the 3-cells that compose a city object without having to iterate through the whole
 177 linear cell complex. This is achieved by introducing the `lccVolumes` attributes in the `CityObject` class,
 178 which contains links to the 3-cells related to the city object. As it is not possible to directly link to the
 179 3-cells, this list contains one of the volume's darts.

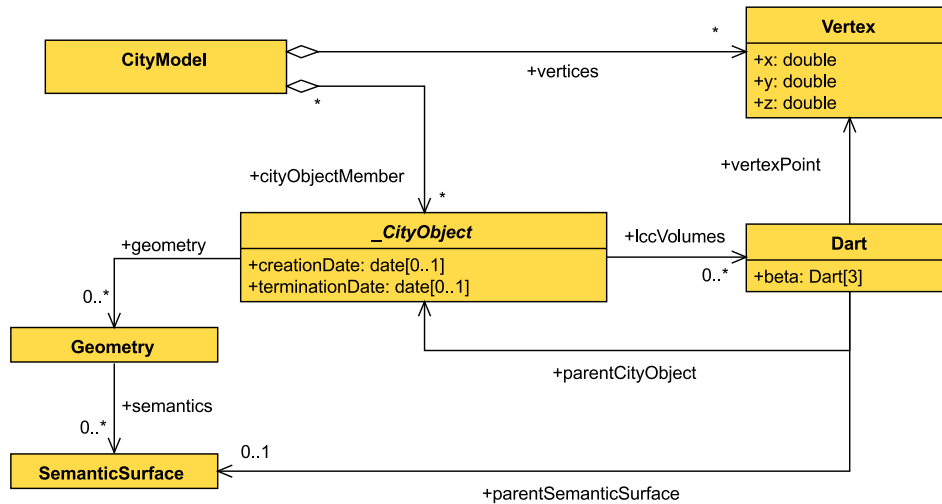


Figure 3. The UML diagram that describes the proposed Linear Cell Complex integration with the CityGML data model.

180 3.2. CityJSON Extension

181 We have, also, developed a CityJSON extension in order to implement the data model as described
 182 in section 3.1. This implementation follows the original principles of the CityJSON data format; aims
 183 to be easy-to-use and compact. For this reason, we have defined two optimisations in the specification
 184 of the extension.

185 First, we reuse the "vertices" list as described in the CityJSON specification. This fits perfectly
 186 with the requirement of associating points to the 0-cells of the C-Map in order to achieve the linear
 187 geometric embedding. Therefore, it is sufficient for the completeness of the final dataset to store
 188 the dart information (their betas and parent object associations) and link them to the existing list of
 189 "vertices", instead of introducing a new list.

190 Second, although in the data model a dart is considered as an object with four attributes ("betas",
 191 "parentCityObject", "semanticSurface" and "vertex"), such a structure would produce a verbose
 192 JSON encoding. That is due to the fact that for every dart in the dataset, the same attribute names
 193 would have to be repeated as keywords. Given the large amount of darts required in order to represent
 194 complicated cell complexes like 3D city models, this would result in a burst of the resulting file's size.
 195 Instead, we decided to store the four attributes as lists with the same length, equal to the number
 196 of darts. This way we avoid the repetition of keywords as they only appear once in the file, thus
 197 minimizing its size.

198 3.2.1. Darts representation

199 In order to store the darts of the LCC we added the new "+darts" root property in the main
 200 "CityJSON" object. It contains four lists containing the values of the respective attributes of the LCCs
 201 darts: "betas", "parentCityObjects", "semanticSurfaces" and "vertices". The "+darts" object
 202 has also the "count" property which states the number of darts in the LCC. These lists are indexed, so
 203 the n -th element of every list corresponds to the respective attribute of the n -th dart of the LCC.⁵

204 According to our implementation, a CityJSON file containing a LCC would contain the following
 205 properties:

```

206 1 {
207 2   "type": "CityJSON",
208 3   "version": "0.9",
  
```

⁵ The lists are one-based numbered, therefore the first element of the list is denoted by the number "1"


```

209 4  "CityObjects": {},
210 5  "vertices": [],
211 6  "+darts": {
212 7    "count": 0,
213 8    "betas": [],
214 9    "parentCityObjects": [],
21510   "semanticSurfaces": [],
21611   "vertices": []
21712 }
21813 "appearance": {},
21914 }

```

220 The "betas" property contains the β_i 's of the darts. Every element of the list is an array of three
 221 integers which refer to the β_1 , β_2 and β_3 of the current dart, respectively. In case this dart is i -free, β_i is
 222 set to -1. Otherwise, the b_i refers to the respective dart's index in the list.

223 The "parentCityObjects" and "vertices" properties are single lists. The first is composed of
 224 the IDs associated with the 2-cells of each dart; and the second is composed of the index of the vertex,
 225 from the CityJSON's "vertices" list, associated with each dart's 0-cell.

226 The "semanticSurfaces" property associates the 2-cell of a dart with a semantic surface of the
 227 city model. Every item of this list is an array of two integer; they refer to the indexes of the geometry
 228 and the semantic surface, respectively, under the parent city object. If the 2-cell of a dart does not have
 229 a semantic surface associated, then the value of the second value is set to -1.

230 The following is an example of a "darts" object that would represent a LCC with one triangle:

```

231 1 "+darts": {
232 2   "betas": [
233 3     [2, -1, -1],
234 4     [3, -1, -1],
235 5     [1, -1, -1]
236 6   ],
237 7   "parentCityObjects": [
238 8     "id-1",
239 9     "id-1",
24010    "id-1"
24111   ],
24212   "semanticSurfaces": [
24313     [0, 0],
24414     [0, 0],
24515     [0, 0]
24616   ],
24717   "vertices": [
24818     0,
24919     1,
25020     2
25121   ]
25222 }

```

253 3.2.2. CityObject to LCC association

254 In order to be able to efficiently identify the 3-cells that compose a city object we added the
 255 "+lccVolumes" property in the "CityObject". The "+lccVolumes" property is a list of darts that
 256 belong to the respective 3-cells. It has to be noted that not all darts related to a city objects are stored in
 257 this list; instead, one dart per 3-cell is used as an index. Therefore, the number of elements in the list
 258 should be equal to the number of volumes associated with the city object.

259 The following is an example of a city object which is associated to three volumes of the LCC.

```

260 1 "CityObjects": {
261 2   "id-1": {
262 3     "type": "Building",
263 4     "attributes": {...},

```

```
264 5     "geometry": [...],  
265 6     "+lccVolumes": [0, 5, 28]  
266 7 }  
267 8 }
```

268 4. Topological reconstruction of 3D city models

269 4.1. Algorithm

270 In order to validate our proposed extension, we have developed an algorithm that parses a
271 CityJSON model and appends the LCC information to the model. In Vitalis *et al.* [12] we have proposed
272 two variations of an algorithm that topologically reconstructs a CityGML model. For the purposes of
273 the research presented in this article, we worked on the base of the “geometric-oriented” algorithm.
274 We chose this variation because it results in a true topological representation of the model. In addition,
275 our proposed extension does not lose semantic information as it associates 2-cells of the resulting LCC
276 with the city model. Therefore, even when multiple city objects are represented under the same 3-cell,
277 the semantic association is retained.

278 We introduced a number of modifications to the original algorithm in order to adjust it towards
279 the requirements of our research. First, the algorithm had to conform with a flat representation of city
280 objects and geometries, as described by the CityJSON specification. Therefore, the recursive call in
281 algorithm 2 have been removed. Second, we only have to define the essential information that ensure
282 a complete association between city objects and cells, as described in 3.2. The resulting methodology is
283 described by algorithms 1, 2, 3, 4, and 5.

284 Initially, the reconstruction is conducted by the main body (algorithm 1). This function iterates
285 through the city objects listed in the city model and calls `ReadCityObject` for every city object.

286 Function `ReadCityObject` (algorithm 2) iterates through the geometries of the city object. For
287 every geometry, function `ReadGeometry` is called and, then, the created darts’ 2-attributes are associated
288 to the object’s id and geometry’s id.

289 The `ReadGeometry` function (algorithm 3) parses the geometry by getting all polygons that bound
290 the object. For every polygon, the algorithm iterates through the edge and created a dart that represents
291 this edge, by calling `GetEdge`. The newly created dart that represents the edge is, then, associated with
292 the semantic surface of the polygon by assigning the respective 2-attribute value. Finally, the algorithm
293 accesses the items of index I_3 in order to find adjacent 3-free 2-cells, in which case the two 2-cells must
294 be 3-sewed.

295 Function `GetEdge` (algorithm 4) creates a dart that represents an edge, given the two end points
296 of the edge. It gets one dart per point by calling the `GetVertex` function and then conducts a 1-sew
297 operation between them. Finally, it iterates through index I_2 in order to find adjacent 2-free 1-cells so
298 that they can be 2-sewed.

299 Finally, function `GetVertex` (algorithm 5) is responsible for creating darts that represent a vertex
300 in the LCC. The function iterates through the LCC in order to find existing 1-free darts with the same
301 coordinates as the point provided; if such a dart is found, it is returned. If no dart is found, the
302 algorithm creates a new dart, associates the coordinates to its 0-attribute, and returns it.

303 4.2. Implementation

304 We have implemented the proposed algorithms in computer software using the C++ programming
305 language. We used the CGAL LCC package⁶ for the data structure that keeps the topological
306 information. JSON for Modern C++⁷ by Niels Lohmann was used for CityJSON.

⁶ https://doc.cgal.org/latest/Linear_cell_complex/index.html

⁷ <https://github.com/nlohmann/json>

Algorithm 1: Main body of reconstruction

Input: city model cm to be processed**Output:** linear cell complex lcc that contains the geometry and semantics of the provided city model

```

1  $R \leftarrow$  Get all root objects of  $cm$ ;
2  $I_2 \leftarrow$  An empty index of darts;
3  $I_3 \leftarrow$  An empty index of darts;
4  $lcc \leftarrow \emptyset$ ;
5 foreach  $obj \in R$  do
6    $\lfloor$  ReadCityObject( $lcc, I_2, I_3, obj$ );
7 return  $lcc$ 

```

Algorithm 2: ReadCityObject

Input: linear cell complex lcc where the geometry of the city object will be added, index I_2 of 2-free darts in lcc , index I_3 of 3-free darts in lcc , city object obj to be processed**Result:** Updates the lcc and I variables according to the provided city object

```

1  $G \leftarrow$  Get all geometries of  $obj$ ;
2  $g_{id} \leftarrow 0$ ;
3 foreach  $g \in G$  do
4    $D \leftarrow$  ReadGeometry( $lcc, I_2, I_3, g$ );
5   foreach  $d \in D$  do
6      $\lfloor$  2-attr-object( $d$ )  $\leftarrow$  id( $obj$ );
7      $\lfloor$  2-attr-geometry( $d$ )  $\leftarrow$   $g_{id}$ ;
8    $g_{id} \leftarrow g_{id} + 1$ ;

```

Algorithm 3: ReadGeometry

Input: linear cell complex lcc where the new 2-cell will be created, index I_2 of 2-free darts in lcc , index I_3 of 3-free darts in lcc , geometry g that a set of polygons**Result:** a new 2-cell is created in lcc and I_2 and I_3 are updated accordingly**Output:** darts D that were used for the creation of the 2-cell

```

1  $D \leftarrow \emptyset$ ;
2  $P \leftarrow$  Get all polygons of  $g$ ;
3 foreach  $poly \in P$  do
4   foreach  $v_{cur} \in poly$  except the last do
5      $v_{next} \leftarrow$  Get next vertex of  $poly$ ;
6      $d_{new} =$  GetEdge( $v_{cur}, v_{next}, I_2$ );
7     2-attr-semantic-surface( $d_{new}$ )  $\leftarrow$  Semantic surface id of  $poly$ ;
8      $\lfloor$  push( $D, d_{new}$ );
9 foreach  $d \in D$  do
10  if  $\exists d_{op} \in I_3$  : reverse key of  $d =$  key of  $d_{op}$  then
11     $\lfloor$  Sew( $d, d_{op}, 3$ );
12     $\lfloor$  Remove  $d_{op}$  from  $I_3$ ;
13  else
14     $\lfloor$  Add  $d$  to  $I_3$ ;
15 return  $D$ ;

```

Algorithm 4: GetEdge

Input: linear cell complex lcc where the new edge belongs
 index I_2 of 2-free darts in the lcc
 vertex v_1 that will be the starting point of the edge
 vertex v_2 that will be the ending point of the edge

Output: dart d_{new} that describes the edge in lcc

```

1  $d_{new} \leftarrow \text{GetVertex}(v_1, 1);$ 
2  $d_{next} \leftarrow \text{GetVertex}(v_2, 0);$ 
3  $\text{Sew}(d_{new}, d_{next}, 1);$ 
4 if  $\exists d_{op} \in I_2 : 0\text{-attr}(d_{op}) = v_2$  and  $0\text{-attr}(\beta_1(d_{op})) = v_1$  then
5   |  $\text{Sew}(d_{new}, d_{op}, 2);$ 
6   | Remove  $d_{op}$  from  $I_2;$ 
7 else
8   | Add  $d_{new}$  to  $I_2;$ 
9 return  $d_{new};$ 

```

Algorithm 5: GetVertex

Input: linear cell complex lcc where the output dart belongs
 vertex v to set of the output dart
 degree of freedom i that the output dart must have

Output: dart d that belongs to lcc , have the vertex v and is i -free

```

1  $D \leftarrow$  Get all darts of  $lcc;$ 
2 foreach  $d \in D$  do
3   | if  $0\text{-attr}(d) = v \wedge \beta_i(d) = \emptyset$  then
4     | return  $d;$ 
5  $d \leftarrow$  Create new dart of  $lcc;$ 
6  $0\text{-attr}(d) \leftarrow v;$ 
7 return  $d;$ 

```

307 The tool created is a command-line application that is available under the MIT license in an public
308 repository⁸. It creates an executable file that can be provided with an existing CityJSON file and create
309 a LCC that represents its geometry. The resulting LCC can be saved either as a CGAL C-Map file
310 (.3map) or as a new CityJSON.

311 5. Validation of methodology

312 In order to verify the completeness of our proposed methodology we have applied it to three
313 open datasets and visualised their topological and semantic information.

314 5.1. Datasets

315 We used the software described in 4.2 in order to reconstruct three existing open dataset available
316 as CityJSON files:

- 317 • Den Haag dataset of buildings and terrain⁹,
- 318 • Rotterdam's Delfshaven dataset of buildings¹⁰, and
- 319 • A dataset representing a landscape around a railway, originally introduced to demonstrate a
320 plethora of CityGML 2.0 city object types.

321 5.1.1. Den Haag

322 This is a dataset of buildings and the terrain provided by the municipality of the Hague. The
323 model was created in 2010 and is based on the aerial photos acquired and the registration of buildings
324 (BAG¹¹) of that year. The dataset concerns around 112.500 buildings of the municipality of the Hague
325 and the neighbouring municipalities, divided in 152 tiles.

326 We tested our methodology against the first tile of the dataset, which is available as example
327 dataset for CityJSON¹². The file contains 2498 city objects, of which one is a TINRelief and the rest are
328 Building objects. It contains 1991 LOD2 geometries of MultiSurface and CompositeSurface type,
329 with semantic surfaces RoofSurface, WallSurface and GroundSurface present.

330 5.1.2. Delfshaven

331 This is the first version of Rotterdam's 3D city model which was released as open data in 2010. It
332 was created based on the basic topographical map of the Dutch Kadaster (BGT¹³) and LiDAR data
333 for the extrusion of the buildings. It is divided in 92 files separated according to the municipalities
334 neighbourhood administration boundaries.

335 In the research described in this article we have worked with the CityJSON file containing the
336 Delfshaven neighbourhood. The file contains 853 building objects of LOD2 and a respective number
337 of MultiSurface geometries with three semantic surfaces present: RoofSurface, WallSurface and
338 GroundSurface.

339 In this dataset the walls between adjacent buildings are missing. This causes multiple buildings
340 to merge under one volume, topologically, instead of being individual volumes one next to the other.

341 5.1.3. Railway demo

342 This datasets is a procedurally produced 3D city model with the intention to demonstrate most
343 of CityGML 2.0 city object types. It is available in CityJSON format and contains 121 city objects of

8 <https://github.com/tudelft3d/cityjson-lcc-reconstructor>

9 <https://data.overheid.nl/dataset/48265-3d-lod2-stadsmodel-2010-den-haag-citygml>

10 <http://rotterdamopendata.nl/dataset/rotterdam-3d-bestanden>

11 <https://zakelijk.kadaster.nl/bag>

12 <http://www.cityjson.org/en/0.9/datasets/>

13 <https://zakelijk.kadaster.nl/bgt>

344 fourteen different types. It, also, contains 105 MultiSurface and CompositeSurface geometries with
 345 semantic surfaces.

346 5.2. LCC Viewer

347 We have built a viewer to evaluate the topologically reconstructed CityJSON files¹⁴. Our viewer's
 348 source code is based on CGAL's demo 3D viewer of the LCC data structure. We added two features
 349 that we needed for our experiments: a) the ability to load CityJSON files with a LCC; and b) an option
 350 to render surfaces, thus objects, in three different ways: per volume, per semantic surface type and per
 351 city object id.

352 5.3. Reconstruction and evaluation of datasets

353 Using our software (section 4.2) we have topologically reconstructed the three datasets and
 354 created three CityJSON files containing the LCC according to the proposed extension (section 3.2). The
 355 characteristics of the resulting datasets is shown in table 1.

Dataset	Filesize (MB)		Number of						
	Original	Final	City objects	Geometries	Darts	0-cells	1-cells	2-cells	3-cells
Den Haag	3,00	8,56	2498	1991	84795	24835	42658	21804	1991
Delfshaven	2,60	7,08	853	853	68500	26782	42644	15484	1192
Railway demo	4,31	18,92	121	105	243491	76821	135514	65196	5789

Table 1. Statistics of the datasets that were reconstructed from our software

356 The resulting datasets have significantly grown in size after the reconstruction. We identified
 357 that the main factor of growth is the number of darts, which seemed to contribute consistently on the
 358 added space of the resulting file. On average, the three datasets required about 65 bytes per dart.

359 In order to verify the complete representation of semantics and cells association we inspected
 360 the final datasets in our viewer (section 5.2). Every dataset was visualised with three different facet
 361 colouring methods: per individual volume, per semantic surface type and per city object id (figure 4).
 362 The per-volume facet formatting highlights the topological characteristics of the dataset. Using the
 363 per-city-object facet formatting we verified that the association between the semantics of the dataset
 364 and the cells of the LCC are retained and complete. Finally, using the per-semantic-surface formatting
 365 we verified that association between surfaces and its semantic information in the respective geometry
 366 of the original model were also maintained.

367 The inspection proved that the proposed CityJSON extension is complete enough to provide the
 368 association between semantics and cells. The viewer successfully highlighted the datasets according to
 369 both topology (volumes) and semantics (city object ids and semantic surfaces).

370 During the inspection of the statistics and the visualisation we identified a degenerate case. We
 371 would have expected the Delfshaven dataset to have less volumes (3-cells) than city objects, given
 372 that multiple buildings were merged during the reconstruction. Nevertheless, that did not occur as,
 373 according to table 1, the number of 3-cells was greater than the number of city objects. After further
 374 investigation of the model, we identified two reasons for this: first, a small number of the 3-cells was
 375 composed by single facets which were topologically invalid with their surroundings, therefore they
 376 weren't merged to the same volume as the rest of the surfaces of those objects; second, a great number
 377 of 3-cells was "noise" in the LCC, as they were single edges without surface or volume (figure 5). We
 378 verified that those edges were present in the initial model as zero-area surfaces and they were retained
 379 in the resulting LCC.

¹⁴ <https://github.com/liberostelios/lcc-viewer>



(a) Every facets (2-cells) is coloured according to the city object in the CityJSON structure. This figure proves that buildings that are merged in the same volume (3-cell) maintain their association with the original city objects.



(b) All facets (2-cells) that are incident to the same volume (3-cell) have the same colour. This figure highlights the topological characteristics of the dataset and shows that multiple semantically individual buildings have been merged, topologically, under the same volume.

(c) Every facet (2-cell) is being coloured according to the semantic surface related to it: *red* highlights roof surfaces and *white* highlight walls. This figure proves that the the resulting dataset maintains the association between facets (2-cells) and semantic surfaces in the CityJSON structure.

Figure 4. The Delfshaven dataset visualised in the LCC viewer according to three different colour formatting options

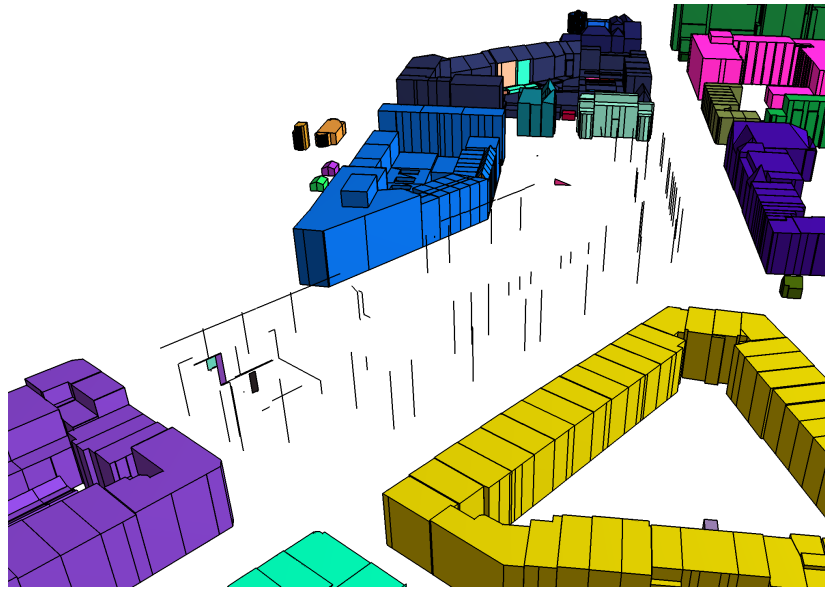


Figure 5. Topologically invalid surfaces and “noisy” single-edged volumes were identified in the Delfshaven LCC when the buildings of the area were hidden.

380 6. Conclusions

381 In this article we propose a topological representation for 3D city models by incorporating a LCC
 382 based on the C-Map data structure in the CityGML data model. We materialized this solution by
 383 developing an extension for CityJSON and the respective algorithms that can compute the topological
 384 links based on the geometry of an existing dataset. Furthermore, we implemented this solution
 385 in a computer software and applied it to three open CityJSON datasets in order to evaluate the
 386 completeness of the proposed solution.

387 Our results show that it is possible to represent any CityGML dataset based on the C-Map data
 388 structure without missing semantic information from the original dataset. In addition, the proposed
 389 two-way linking mechanism between the entities of the data model and the LCC, provides access
 390 to the efficient resulting 3D city model based on either a semantic-oriented traversal—by iterating
 391 through every city object of the model—or a geometric-oriented traversal—by visiting all darts of the
 392 LCC—.

393 We believe that our solution can provide useful information for the geometrical processing of 3D
 394 city models. For example, it can assist on the repair of invalid geometries, such as non-watertight solid,
 395 based on the existence of 2-free darts in the LCC. In addition, our findings regarding “noisy” 3-cells
 396 in the Delfshaven dataset (section 5.3) proves that LCC statistics can provide useful insights for the
 397 identification of invalid or erroneous data. In the future, we intend to utilise this CityJSON extension
 398 in order to conduct analysis on the topological matching of existing multi-LoD datasets. We are, also,
 399 planning to use the proposed data structure in order to represent those datasets in four dimensions.

400 **Funding:** This project has received funding from the European Research Council (ERC) under the European
 401 Union’s Horizon 2020 research and innovation programme (grant agreement No 677312 UMnD).

402 **Acknowledgments:** We would like to thank Tom Commandeur for his assistance in the optimisation of our
 403 topological reconstruction software and Kavisha for her insights in the design of UML diagrams describing our
 404 data model.

405 Abbreviations

406 The following abbreviations are used in this manuscript:

407

C-Map	combinatorial map
LCC	linear cell complex
GML	geography markup language
408 SFS	simple feature specification
BIM	building information modelling
GIS	geographic information system
JSON	JavaScript Object Notation

409

- 410 1. Choi, J.; Lee, J., 3D Geo-Network for Agent-based Building Evacuation Simulation. In *3D Geo-Information*
411 *Sciences*; Lee, J.; Zlatanova, S., Eds.; Springer Berlin Heidelberg: Berlin, Heidelberg, 2009; pp. 283–299.
412 doi:10.1007/978-3-540-87395-2_18.
- 413 2. Strand, R.; Baumgartner, K. Modeling radiant heating and cooling systems: integration
414 with a whole-building simulation program. *Energy and Buildings* **2005**, *37*, 389–397.
415 doi:10.1016/j.enbuild.2004.07.009.
- 416 3. Zhivov, A.M.; Case, M.P.; Jank, R.; Eicker, U.; Booth, S. Planning Tools to Simulate
417 and Optimize Neighborhood Energy Systems. In *Green Defense Technology*; Springer, 2017.
418 doi:10.1007/978-94-017-7600-4_8.
- 419 4. Open Geospatial Consortium. City Geography Markup Language (CityGML) Encoding Standard, version:
420 2.0.0, 2012.
- 421 5. Guo, Y.; Pan, M.; Wang, Z.; Qu, H.; Lan, X. A spatial overlay analysis method for three-dimensional
422 vector polyhedrons. 2010 18th International Conference on Geoinformatics. IEEE, 2010.
423 doi:10.1109/geoinformatics.2010.5567674.
- 424 6. Maria, M.; Horna, S.; Aveneau, L. Topological Space Partition for Fast Ray Tracing in Architectural Models.
425 GRAPP 2014 - 9th International Joint Conference on Computer Graphics Theory and Applications; , 2014;
426 pp. 225 – 235.
- 427 7. Arroyo Otori, K.; Ledoux, H.; Stoter, J. An evaluation and classification of nD topological data structures
428 for the representation of objects in a higher-dimensional GIS. *International Journal of Geographical Information*
429 *Science* **2015**, *29*, 825–849. doi:10.1080/13658816.2014.999683.
- 430 8. Arroyo Otori, K.; Ledoux, H.; Stoter, J. A dimension-independent extrusion algorithm using
431 generalised maps. *International Journal of Geographical Information Science* **2015**, *29*, 1166–1186.
432 doi:10.1080/13658816.2015.1010535.
- 433 9. Damiand, G.; Teillaud, M. A Generic Implementation of dD Combinatorial Maps in CGAL. *Procedia*
434 *Engineering* **2014**, *82*, 46–58. doi:10.1016/j.proeng.2014.10.372.
- 435 10. Diakité, A.A.; Damiand, G.; Van Maercke, D. Topological Reconstruction of Complex 3D Buildings
436 and Automatic Extraction of Levels of Detail. Eurographics Workshop on Urban Data Modelling and
437 Visualisation; Gonzalo Besuievsky, V.T., Ed.; Eurographics Association: Strasbourg, France, 2014; pp. 25–30.
438 doi:10.2312/udmv.20141074.
- 439 11. Diakité, A.A.; Damiand, G.; Gesquière, G. Automatic Semantic Labelling of 3D Buildings Based on
440 Geometric and Topological Information. Proc. of 9th International 3DGeoInfo Conference (3DGeoInfo);
441 Karlsruhe Institute of Technology: Dubai, United Arab Emirates, 2014; 3DGeoInfo conference proceedings
442 series.
- 443 12. Vitalis, S.; Otori, K.A.; Stoter, J. Topological Reconstruction of 3D City Models with preservation
444 of semantics. Geospatial Technologies for All : short papers, posters and poster abstracts
445 of the 21th AGILE Conference on Geographic Information Science; Mansourian, A.; Pilesjö,
446 P.; Harrie, L.; von Lammeren, R., Eds. Lund University, 2018. Accessible through
447 <https://agile-online.org/index.php/conference/proceedings/proceedings-2018>.
- 448 13. Open Geospatial Consortium. OpenGIS Geography Markup Language (GML) Encoding Specification,
449 version: 3.1.1, 2012.
- 450 14. Damiand, G.; Lienhardt, P. *Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image*
451 *Processing*, 1st ed.; A. K. Peters, Ltd.: Natick, MA, USA, 2014.