Article

Incorporating topological representation in 3D City Models

Stelios Vitalis *,‡ Ken Arroyo Ohori ‡ 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

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 May 2019 doi:10.20944/preprints201905.0024.v1

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

For this reason, we have investigated the use of ordered topological structures and, more specifically, combinatorial maps (C-Maps) as an alternative to the SFS for the representation of geometric information in 3D city models. C-Maps combine the powerful algebra of geometric simplicial complexes with the ease of construction of polygonal modelling [7]. Regarding the practical aspect, they are implemented in a software package as part of CGAL [1] and they are efficient with respect to memory usage [9]. While C-Maps originally store only topological relationships between objects, they can be easily enhanced with the association of coordinates to vertices, which results in a linear cell complex (LCC) that incorporate both geometric and topological information.

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

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

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

[1] The Computational Geometry Algorithms Library (http://www.cgal.org)

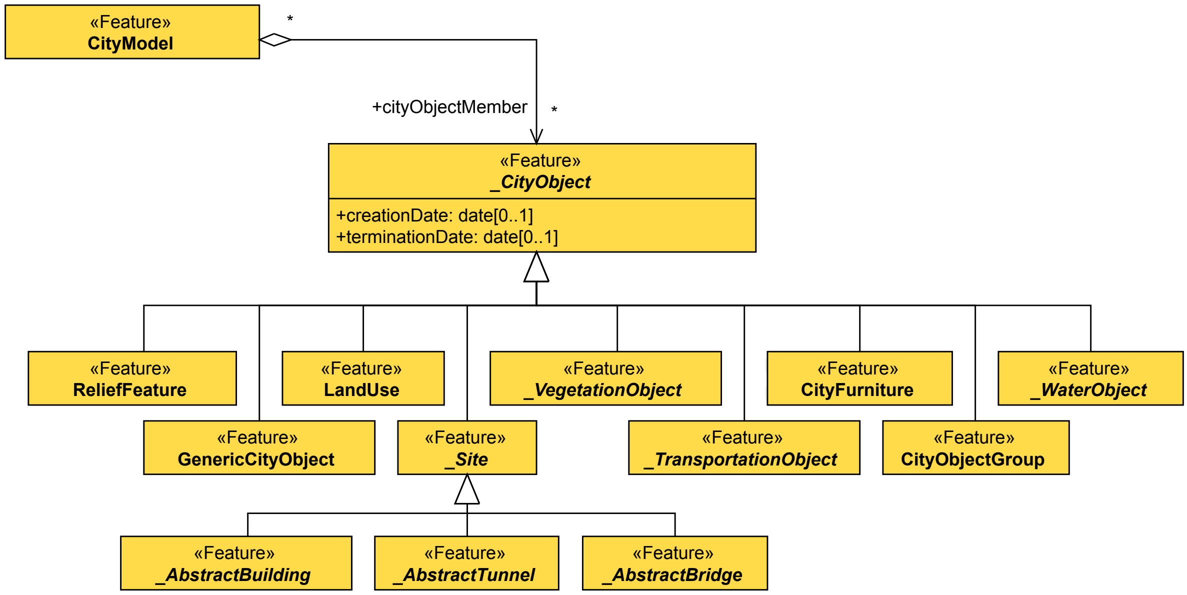


Figure 1. The UML diagram that describes CityGML's top level class hierarchy of city objects.

2. Related work and background information

2.1. The CityGML data model

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

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

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

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

2.2. The CityJSON data format

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

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 May 2019 doi:10.20944/preprints201905.0024.v1

data format, which is verbose and complex due to its GML (and, thus, XML) nature. For example, there are at least 26 different ways to encode a simple four-points' square in GML [2].

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

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

2.3. Combinatorial Maps

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

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

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

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

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

3. Topological representation of 3D city models

In order to represent the topological information of city objects in a CityJSON file we followed two steps: first, we introduced the LCC entities into the CityGML data model; and second, we developed a CityJSON extension that provides the necessary encoding instruction in order to store those information in a CityJSON file. We have developed the extension definition according to the respective CityJSON specification[3]. The CityJSON extension definition is available as open source in GitHub[4].

[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

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 May 2019 doi:10.20944/preprints201905.0024.v1

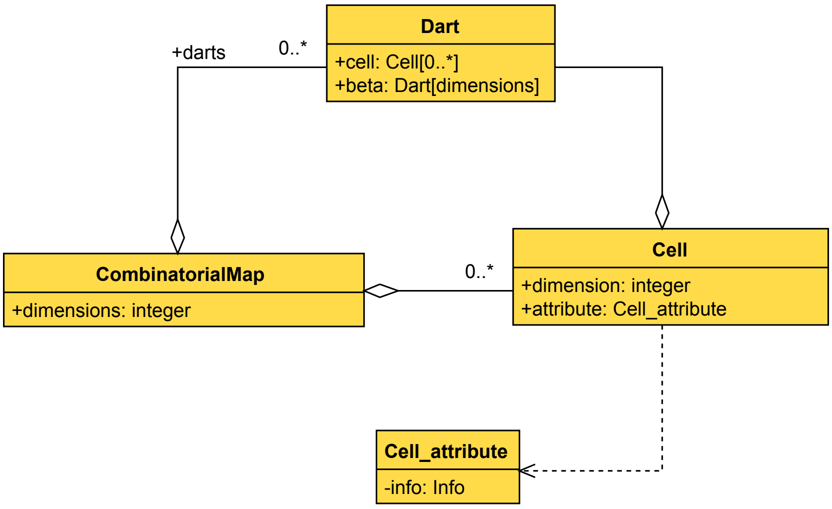


Figure 2. The UML diagram that describes the Linear Cell Complex implementation of CGAL

3.1. Data model

First, it must be possible to store darts in the city model. Second, every dart has to be linked with: (a) the point that defines the location of the dart's 0-cell, (b) the city object that is associated to the dart's 2-cell, (c) the semantic information associated with the dart's 2-cell, and (d) its β_i darts. It 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 structure. Therefore, for a three-dimensional LCC β_1, β_2, and β_3 are only required to be stored.

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

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

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

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

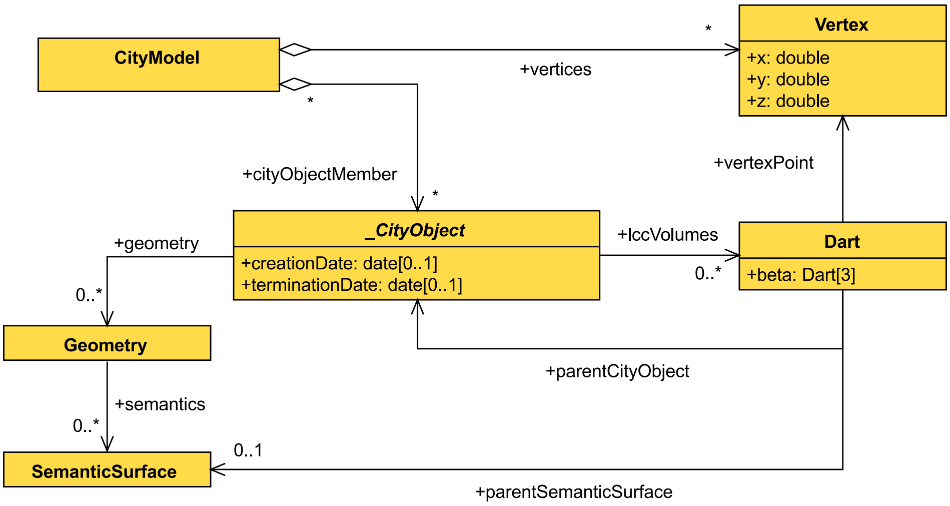


Figure 3. The UML diagram that describes the proposed Linear Cell Complex integration with the CityGML data model.

3.2. CityJSON Extension

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

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

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

3.2.1. Darts representation

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

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

```
{
  "type": "CityJSON",
  "version": "0.9",
```

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

```
  "CityObjects": {},
  "vertices": [],
  "+darts": {
    "count": 0,
    "betas": [],
    "parentCityObjects": [],
    "semanticSurfaces": [],
    "vertices": []
  }
  "appearance": {},
}
```

The "betas" property contains the β_i's of the darts. Every element of the list is an array of three 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 set to -1. Otherwise, the b_i refers to the respective dart's index in the list.

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

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

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

```
"+darts": {
  "betas": [
    [2, -1, -1],
    [3, -1, -1],
    [1, -1, -1]
  ],
  "parentCityObjects": [
    "id-1",
    "id-1",
    "id-1"
  ],
  "semanticSurfaces": [
    [0, 0],
    [0, 0],
    [0, 0]
  ],
  "vertices": [
    0,
    1,
    2
  ]
}
```

3.2.2. CityObject to LCC association

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

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

```
"CityObjects": {
  "id-1": {
    "type": "Building",
    "attributes": {...},
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 May 2019 doi:10.20944/preprints201905.0024.v1


```
    "geometry": [...],
    "+lccVolumes": [0, 5, 28]
  }
}
```

4. Topological reconstruction of 3D city models

4.1. Algorithm

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

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

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

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

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

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

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

4.2. Implementation

We have implemented the proposed algorithms in computer software using the C++ programming language. We used the CGAL LCC package[6] for the data structure that keeps the topological 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

 doi:10.20944/preprints201905.0024.v1

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

$R \leftarrow$ Get all root objects of *cm*;
$I_2 \leftarrow$ An empty index of darts;
$I_3 \leftarrow$ An empty index of darts;
$lcc \leftarrow \emptyset$;
foreach $obj \in R$ **do**
    `ReadCityObject`(lcc, I_2, I_3, obj);
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

$G \leftarrow$ Get all geometries of *obj*;
$g_{id} \leftarrow 0$;
foreach $g \in G$ **do**
    $D \leftarrow$ `ReadGeometry`(lcc, I_2, I_3, g);
    foreach $d \in D$ **do**
        `2-attr-object`(d) $\leftarrow$ `id`(obj);
        `2-attr-geometry`(d) $\leftarrow g_{id}$;
    $g_{id} \leftarrow g_{id} + 1$;

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

$D \leftarrow \emptyset$;
$P \leftarrow$ Get all polygons of *g*;
foreach $poly \in P$ **do**
    foreach $v_{cur} \in poly$ except the last **do**
        $v_{next} \leftarrow$ Get next vertex of *poly*;
        d_{new} = `GetEdge`(v_{cur}, v_{next}, I_2);
        `2-attr-semantic-surface`(d_{new}) $\leftarrow$ Semantic surface id of *poly*;
        `push`(D, d_{new});
foreach $d \in D$ **do**
    if $\exists d_{op} \in I_3$: *reverse key of d = key of* d_{op} **then**
        `Sew`($d, d_{op}, 3$);
        Remove d_{op} from I_3;
    else
        Add d to I_3;
return D;

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 May 2019 doi:10.20944/preprints201905.0024.v1

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*

$d_{new} \leftarrow$ `GetVertex`$(v_1, 1)$;
$d_{next} \leftarrow$ `GetVertex`$(v_2, 0)$;
`Sew`$(d_{new}, d_{next}, 1)$;
if $\exists d_{op} \in I_2$: *0-attr*$(d_{op}) = v_2$ ***and*** *0-attr*$(\beta_1(d_{op}) = v_1)$ **then**
  `Sew`$(d_{new}, d_{op}, 2)$;
  Remove d_{op} from I_2;
else
  Add d_{new} to I_2;
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

$D \leftarrow$ Get all darts of *lcc*;
foreach $d \in D$ **do**
  if `0-attr`$(d) = v \wedge \beta_i(d) = \varnothing$ **then**
    return d;
$d \leftarrow$ Create new dart of *lcc*;
`0-attr`$(d) \leftarrow v$;
return d;

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 May 2019 doi:10.20944/preprints201905.0024.v1

The tool created is a command-line application that is available under the MIT license in an public repository[8]. It creates an executable file that can be provided with an existing CityJSON file and create a LCC that represents its geometry. The resulting LCC can be saved either as a CGAL C-Map file (`.3map`) or as a new CityJSON.

5. Validation of methodology

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

5.1. Datasets

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

- Den Haag dataset of buildings and terrain[9],
- Rotterdam's Delfshaven dataset of buildings[10], and
- A dataset representing a landscape around a railway, originally introduced to demonstrate a plethora of CityGML 2.0 city object types.

5.1.1. Den Haag

This is a dataset of buildings and the terrain provided by the municipality of the Hague. The model was created in 2010 and is based on the aerial photos acquired and the registration of buildings (BAG[11]) of that year. The dataset concerns around 112.500 buildings of the municipality of the Hague and the neighbouring municipalities, divided in 152 tiles.

We tested our methodology against the first tile of the dataset, which is available as example dataset for CityJSON[12]. The file contains 2498 city objects, of which one is a `TINRelief` and the rest are `Building` objects. It contains 1991 LOD2 geometries of `MultiSurface` and `CompositeSurface` type, with semantic surfaces `RoofSurface`, `WallSurface` and `GroundSurface` present.

5.1.2. Delfshaven

This is the first version of Rotterdam's 3D city model which was released as open data in 2010. It was created based on the basic topolographical map of the Dutch Kadaster (BGT[13]) and LiDAR data for the extrusion of the buildings. It is divided in 92 files separated according to the municipalities neighbourhood administration boundaries.

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

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

5.1.3. Railway demo

This datasets is a procedurally produced 3D city model with the intention to demonstrate most 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

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 May 2019 doi:10.20944/preprints201905.0024.v1

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

5.2. LCC Viewer

We have built a viewer to evaluate the topologically reconstructed CityJSON files[14]. Our viewer's source code is based on CGAL's demo 3D viewer of the LCC data structure. We added two features that we needed for our experiments: a) the ability to load CityJSON files with a LCC; and b) an option to render surfaces, thus objects, in three different ways: per volume, per semantic surface type and per city object id.

5.3. Reconstruction and evaluation of datasets

Using our software (section 4.2) we have topologically reconstructed the three datasets and created three CityJSON files containing the LCC according to the proposed extension (section 3.2). The 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

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

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

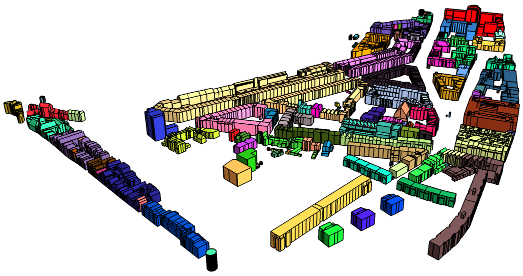
The inspection proved that the proposed CityJSON extension is complete enough to provide the association between semantics and cells. The viewer successfully highlighted the datasets according to both topology (volumes) and semantics (city object ids and semantic surfaces).

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

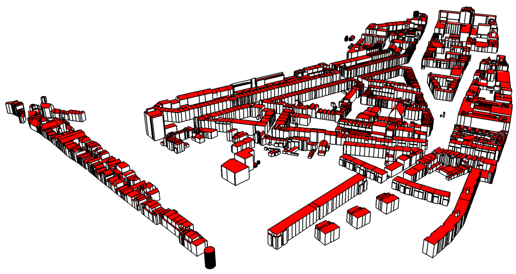
[14] https://github.com/liberostelios/lcc-viewer

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 May 2019 doi:10.20944/preprints201905.0024.v1

(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

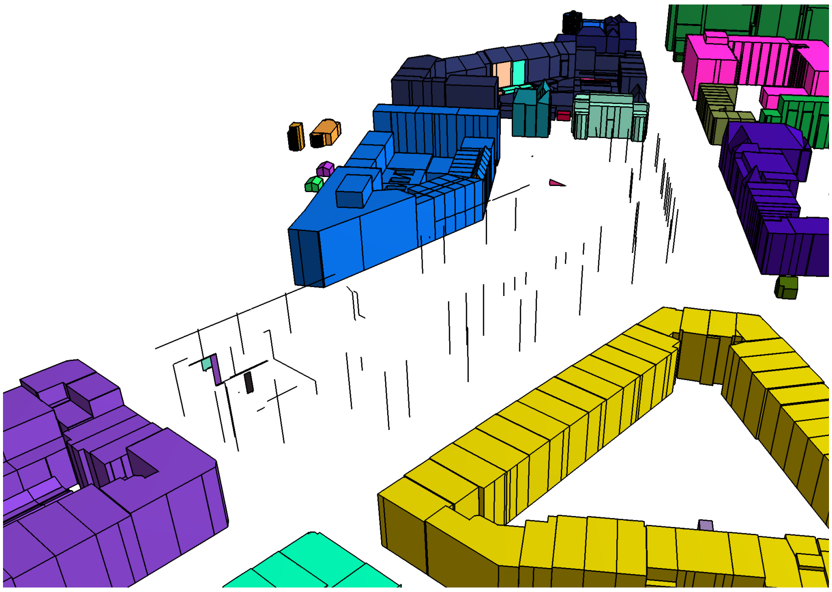


Figure 5. Topologically invalid surfaces and "noisy" single-edged volumes were identified in the Delfshaven LCC when the buildings of the area where hidden.

6. Conclusions

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

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

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

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

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

Abbreviations

The following abbreviations are used in this manuscript:

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

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