

Lesson 4.2

Semantic 3D city models

GEO1004:
3D modelling of the built environment

<https://3d.bk.tudelft.nl/courses/geo1004>



3D geoinformation

Department of Urbanism
Faculty of Architecture and the Built Environment
Delft University of Technology

JSON

JSON

- **JavaScript Object Notation**
- Lightweight data-interchange format
- It is easy for humans to read and write
- It is text-based
- It is easy for machines to parse and generate
- It is originally based on a subset of JavaScript (but is now in most languages)

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```


Virtually all languages have native support for it

It is built on two (very common) **data structures**:

1. collection of name-value pairs
 - Python dictionaries (`a={}`)
 - C++ `std::map`
 - also called a “hash” or “hashmap”
2. arrays
 - Python lists (`a=[]`)
 - C++ `std::vector` / `std::array`

Data types:

- number (float and double)
- integer
- string
- Boolean
- null
- *(no support for datetime type)*

To represent a person

```
{  
  "firstName": "John",  
  "lastName": "Smith",  
  "isAlive": true,  
  "age": 27,  
  "address": {  
    "streetAddress": "21 2nd Street",  
    "city": "New York",  
    "state": "NY",  
    "postalCode": "10021-3100"  
  },  
  "phoneNumbers": [  
    {  
      "type": "home",  
      "number": "212 555-1234"  
    },  
    {  
      "type": "office",  
      "number": "646 555-4567"  
    },  
    {  
      "type": "mobile",  
      "number": "123 456-7890"  
    }  
  ],  
  "children": [],  
  "spouse": null  
}
```

Key-value pair

"name": "Hugo"

A CityJSON file

```
{
  "type": "CityJSON",
  "version": "1.1",
  "metadata": {
    "referenceSystem": "https://www.opengis.net/def/crs/EPSG/0/7415"
  },
  "transform": {...}
  "CityObjects": {
    "id-1": {
      "type": "Building",
      "attributes": {
        "measuredHeight": 22.3,
        "roofType": "gable",
        "owner": "Elvis Presley"
      },
      "geometry": [
        {
          "type": "MultiSurface",
          "boundaries": [
            [[0, 3, 2, 1]], [[4, 5, 6, 7]], [[0, 1, 5, 4]]
          ]
        }
      ]
    }
  },
  "vertices": [
    [231, 23212, 110],
    [1111, 3211, 120],
    ...
  ],
  "appearance": {
    "materials": [],
    "textures": [],
    "vertices-texture": []
  }
}
```

XML (eXtensible Markup Language)

- a markup language that defines a set of rules for encoding information and documents
- both human-readable and machine-readable
- designed to store and transport data
- it is self-descriptive (what data “is”)
- designed to focus on documents, but widely used for the representation of arbitrary data structures
- hierarchical structure (a tree)
- examples of XML? ...

Sources:

<https://en.wikipedia.org/wiki/XML>

https://www.w3schools.com/xml/xml_what.asp

XML (eXtensible Markup Language)

- a markup language that defines a set of rules for encoding information and documents
- both human-readable and machine-readable
- designed to store and transport data
- it is self-descriptive (what data “is”)
- designed to focus on documents, but widely used for the representation of arbitrary data structures
- hierarchical structure (a tree)
- examples of XML? HTML, GML, KML, DOCX, RSS, SVG, COLLADA, GPX, etc.

Sources:

<https://en.wikipedia.org/wiki/XML>

https://www.w3schools.com/xml/xml_what_is.asp

XML (eXtensible Markup Language)

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- no predefined tags
- tags are <note>, <from>, etc.
- authors must define both the tags and the document structure, the latter is done with a *schema*

Sources:

<https://en.wikipedia.org/wiki/XML>

https://www.w3schools.com/xml/xml_what_is.asp

XML elements (or nodes)

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- start-tag: <from>
- end-tag: </from>
- empty-element tag: <from/>
- element can be either:
 - string
 - another element (hierarchy)

Sources:

<https://en.wikipedia.org/wiki/XML>

https://www.w3schools.com/xml/xml_what_is.asp

XML attributes



```
<person gender="female">  
  <firstname>Anna</firstname>  
  <lastname>Smith</lastname>  
</person>
```

```
<person>  
  <gender>female</gender>  
  <firstname>Anna</firstname>  
  <lastname>Smith</lastname>  
</person>
```

Sources:

<https://en.wikipedia.org/wiki/XML>

https://www.w3schools.com/xml/xml_what_is.asp

XML == tree structure

A single root element contains all the other elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

Sources:

<https://en.wikipedia.org/wiki/XML>

https://www.w3schools.com/xml/xml_what.asp

XML: namespace to avoid conflicts

A simple
element



```
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="https://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

Sources:

<https://en.wikipedia.org/wiki/XML>

https://www.w3schools.com/xml/xml_what_is.asp

XML: schemas describe the valid/allowed structure

XSD = XML Schema Definition

```
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Sources:

<https://en.wikipedia.org/wiki/XML>

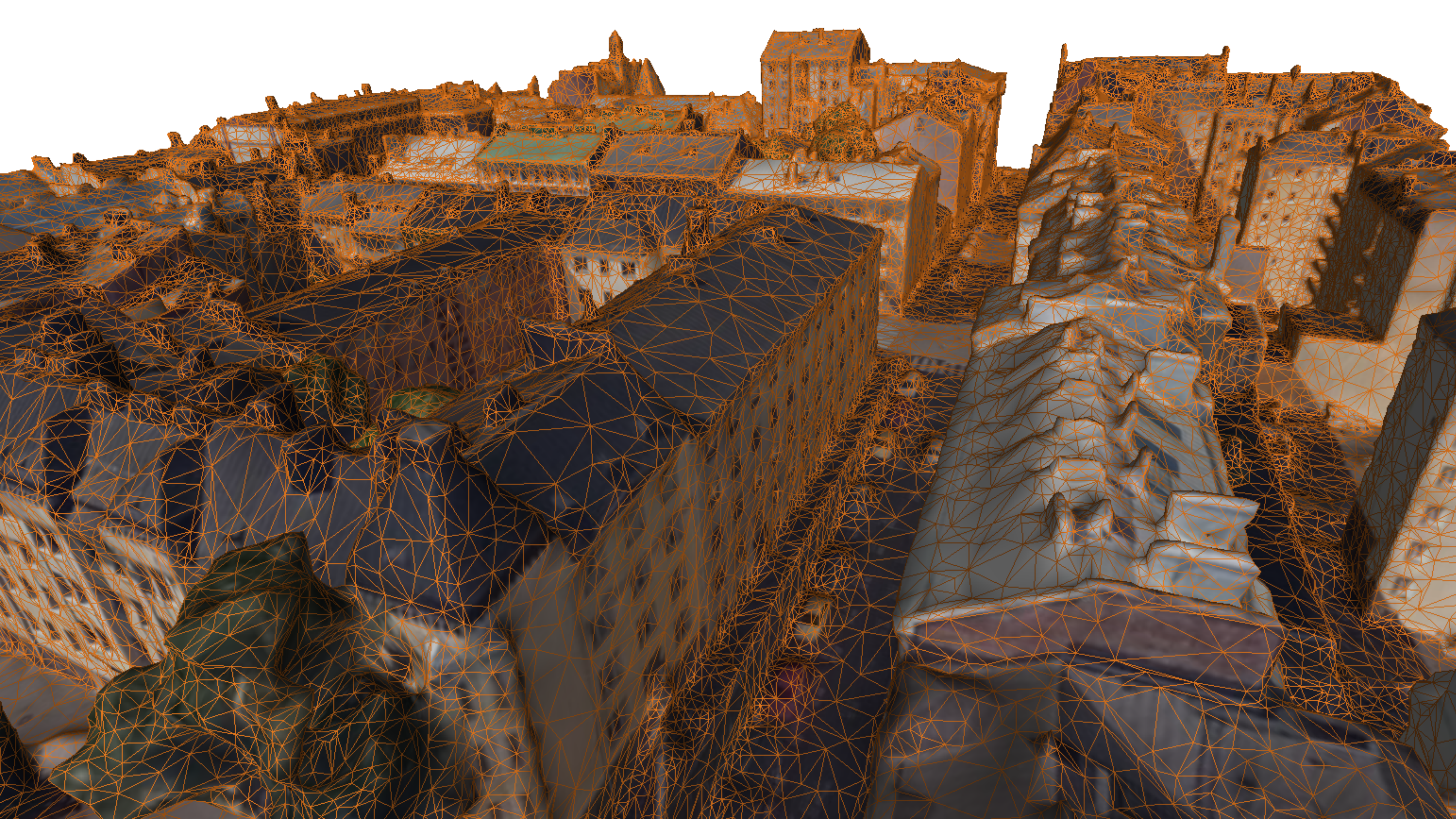
https://www.w3schools.com/xml/xml_what_is.asp

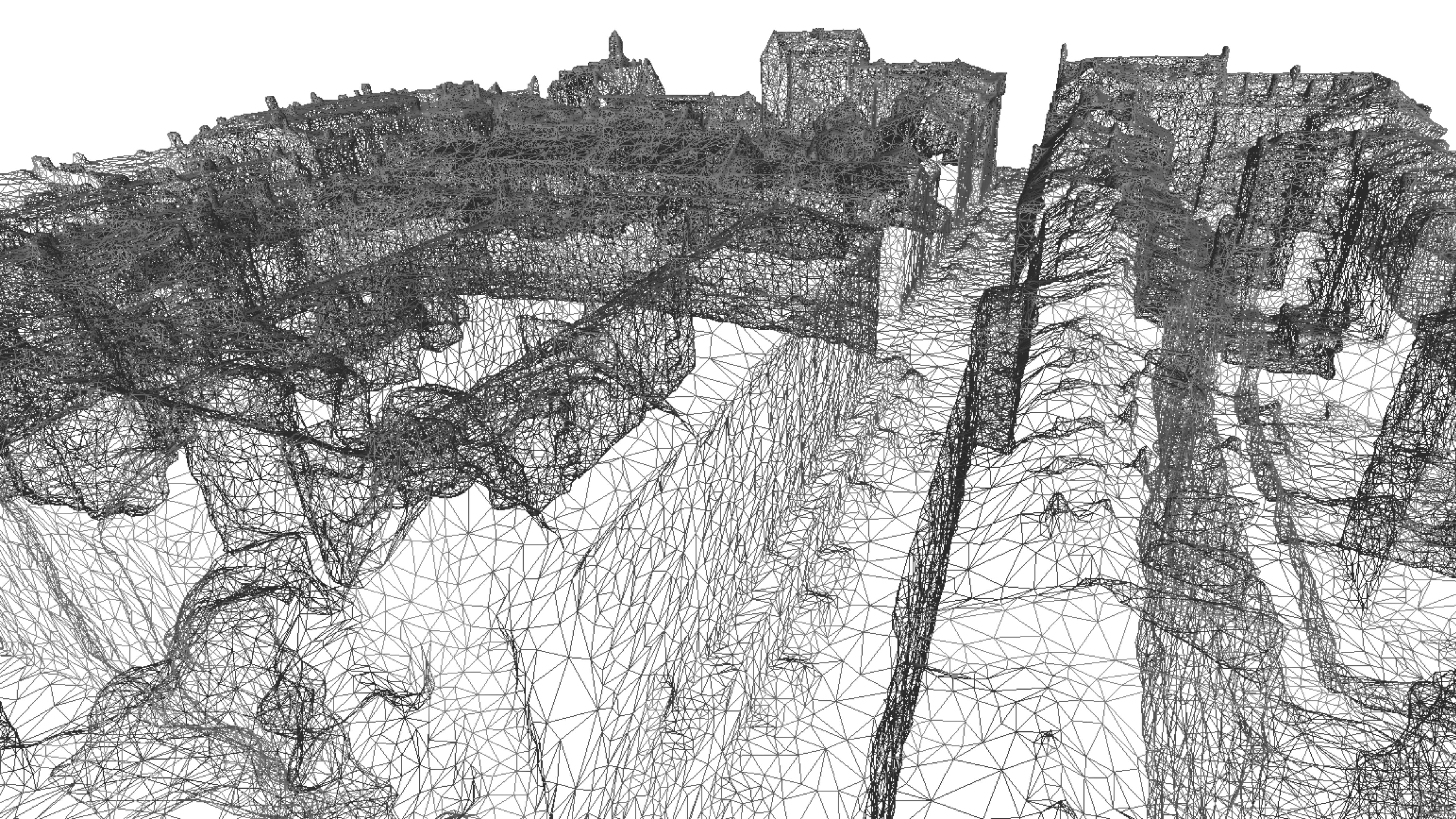
City/GML/JSON

A 3D city model

- a **digital representation**, with 3D geometries, of the common objects in an urban environment,
- **buildings** usually being the most prominent objects.
- typical reconstructed/derived from **various acquisition techniques**:
 - photogrammetry
 - laser scanning
 - extrusion from 2D footprints
 - conversion from architectural models and drawings
 - procedural modelling
 - volunteered geoinformation





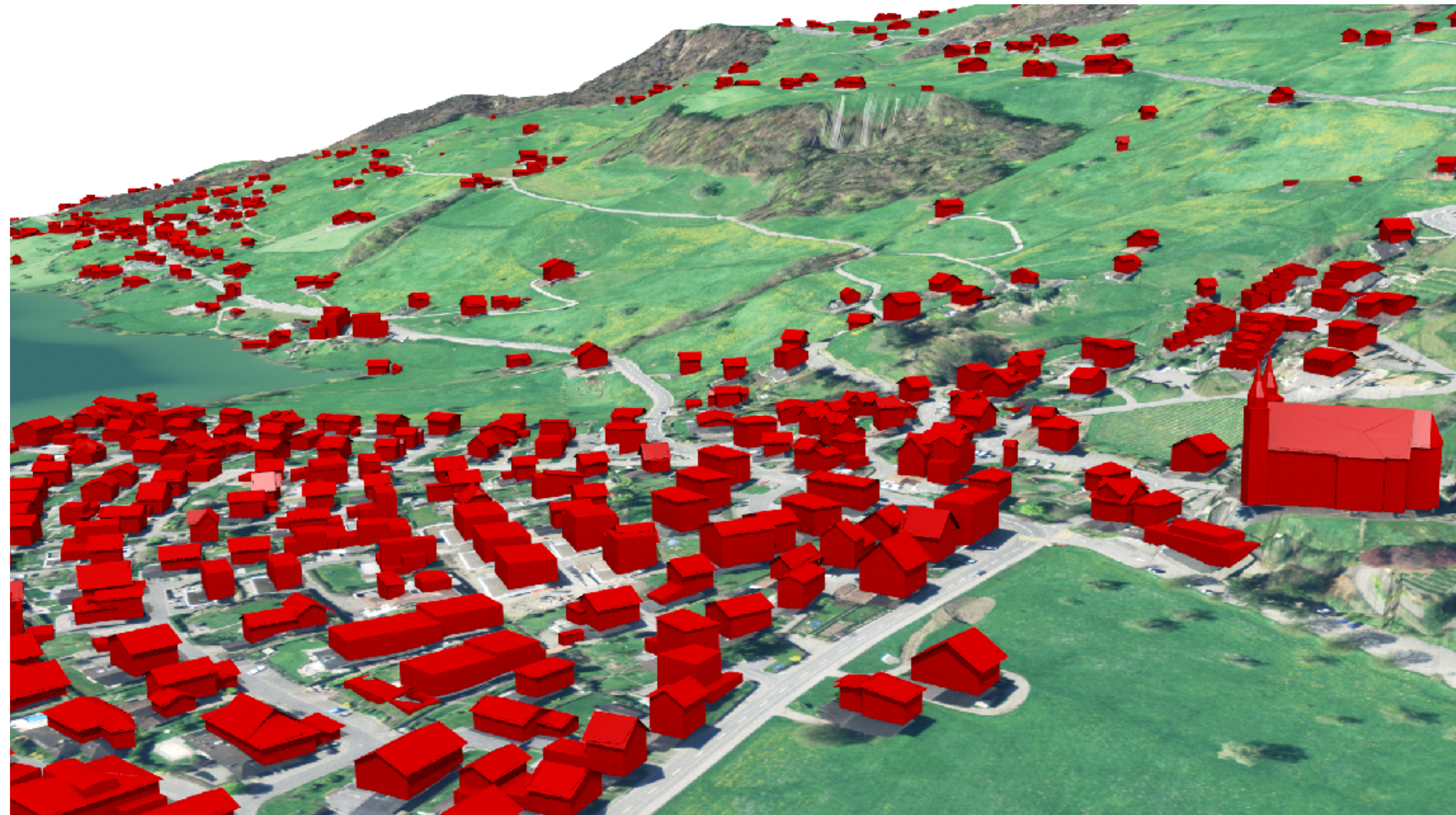
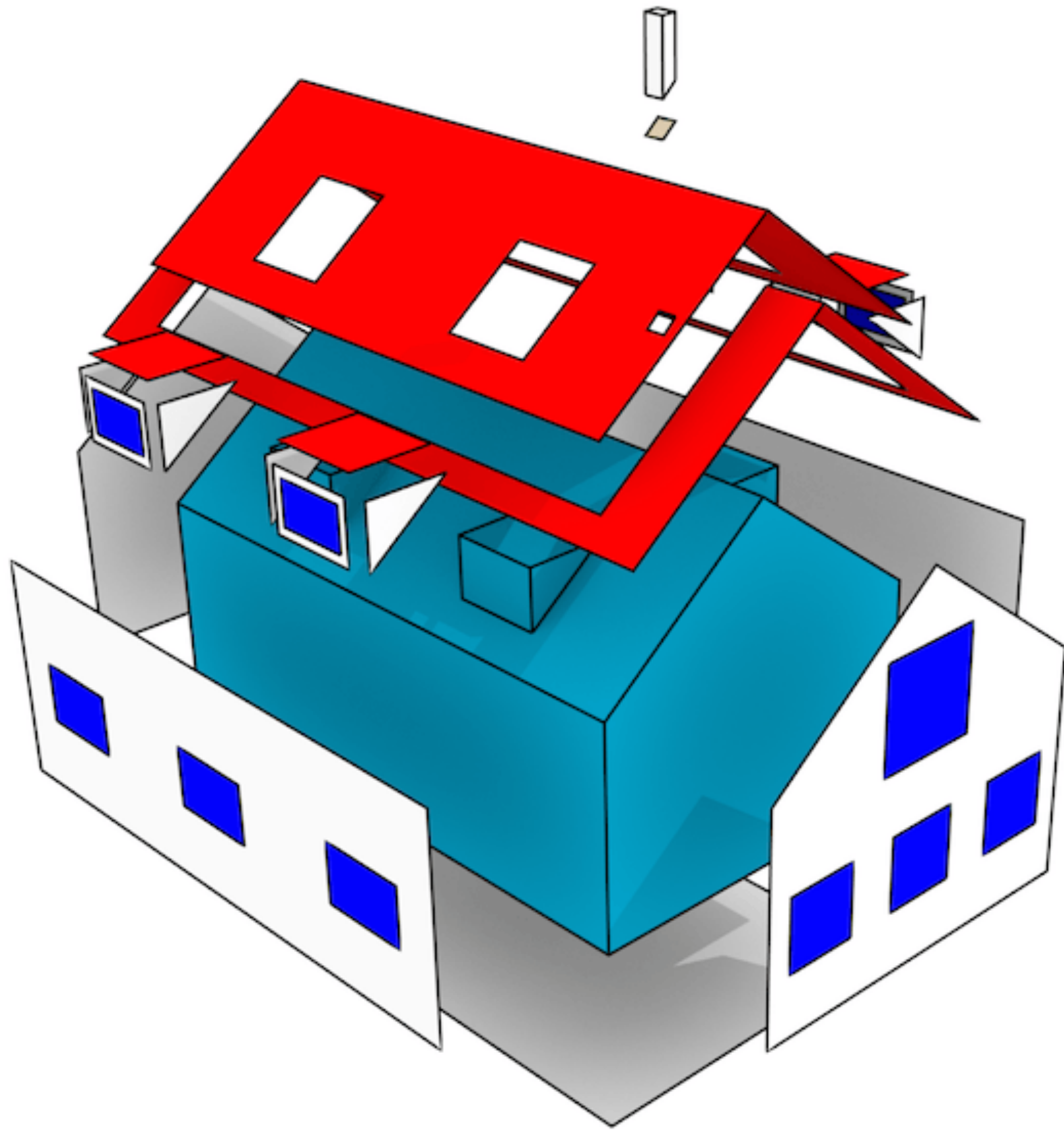


Semantic 3D city model

- data model where the **relevant** objects (and their sub-parts) are labelled with their meaning and have attributes attached to them.
- a city is **decomposed into classes** that we deem relevant for certain applications,
- for instance: 'building', 'road', 'tree', 'lamppost', etc
- The decomposition is thus **hierarchical**, and the relationships between the classes are stored

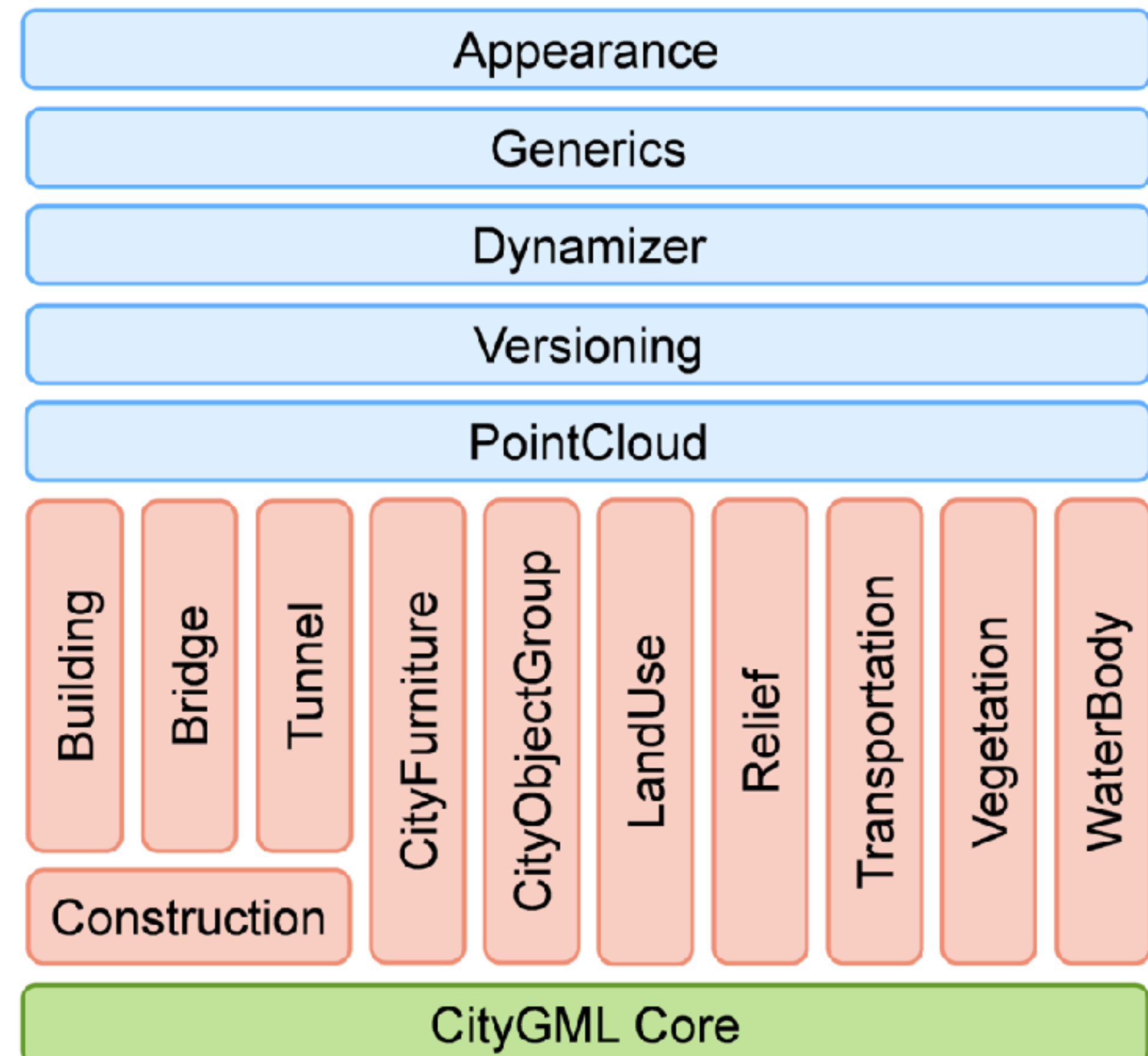
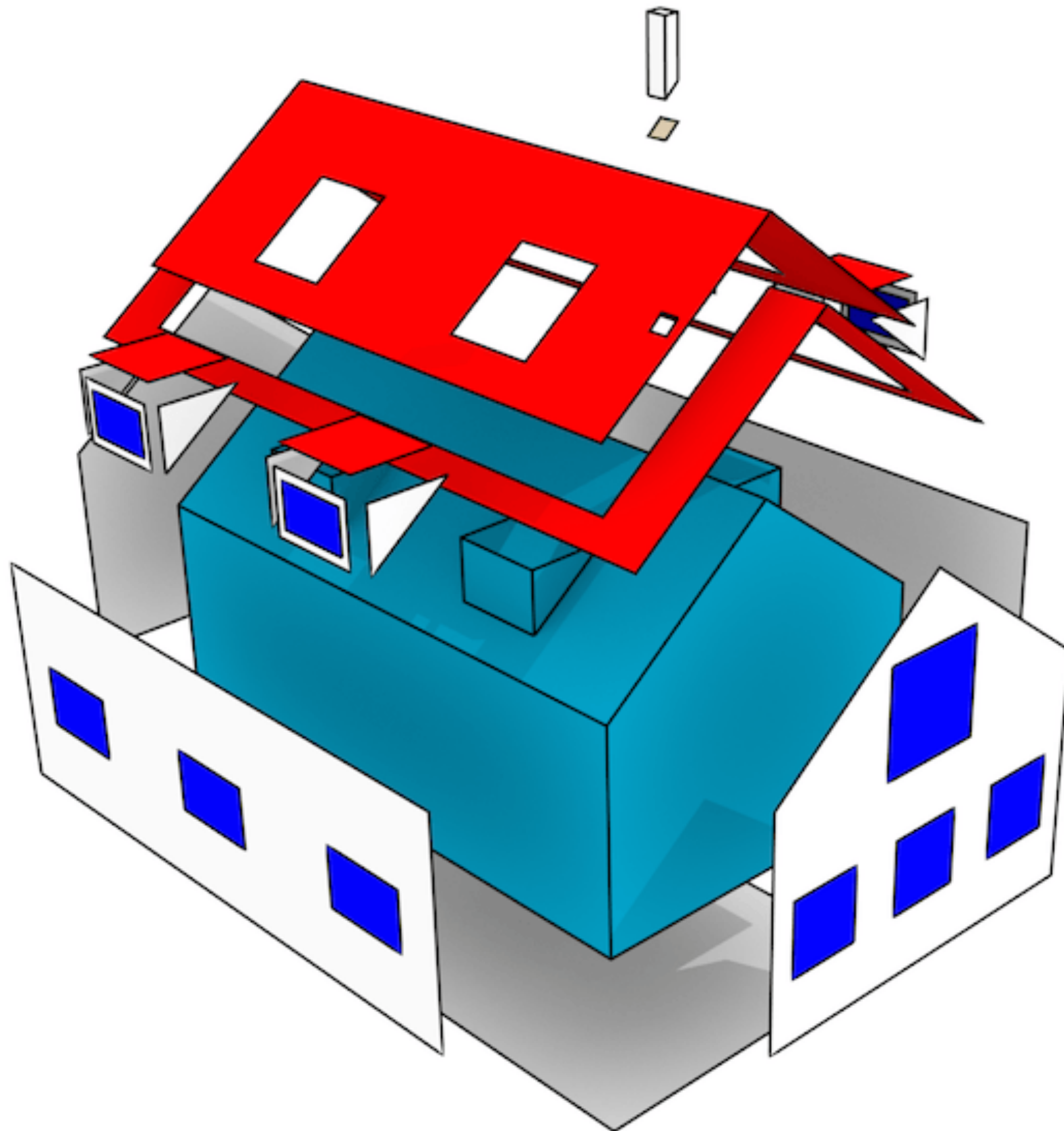


- International standard (from OGC) for representing and storing 3D city models
- Currently at version 3.0
- Both a data model & an encoding (confusing, huh?)





- International standard (from OGC) for representing and storing 3D city models
- Currently at version 3.0
- Both a data model & an encoding (confusing, huh?)



Spatio-semantic coherence

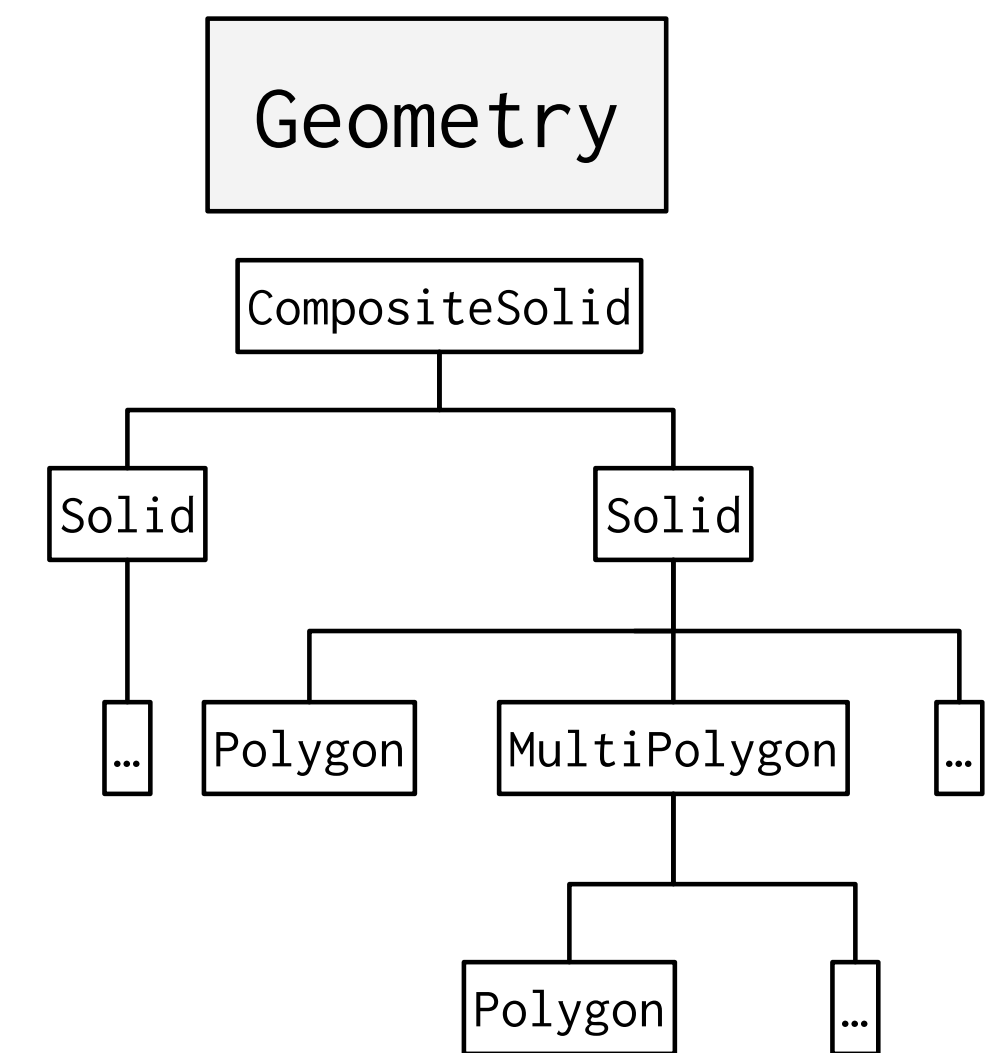
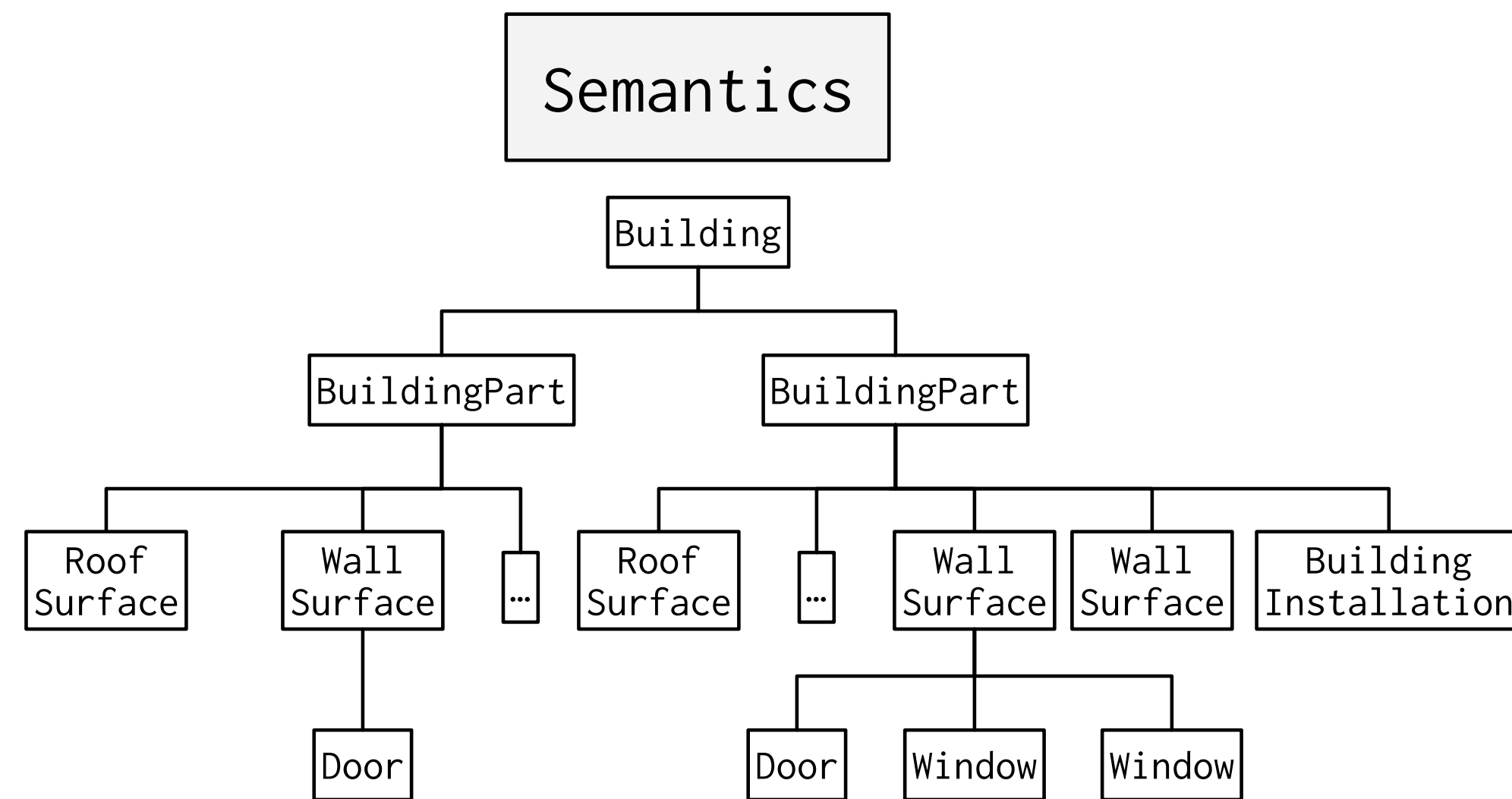
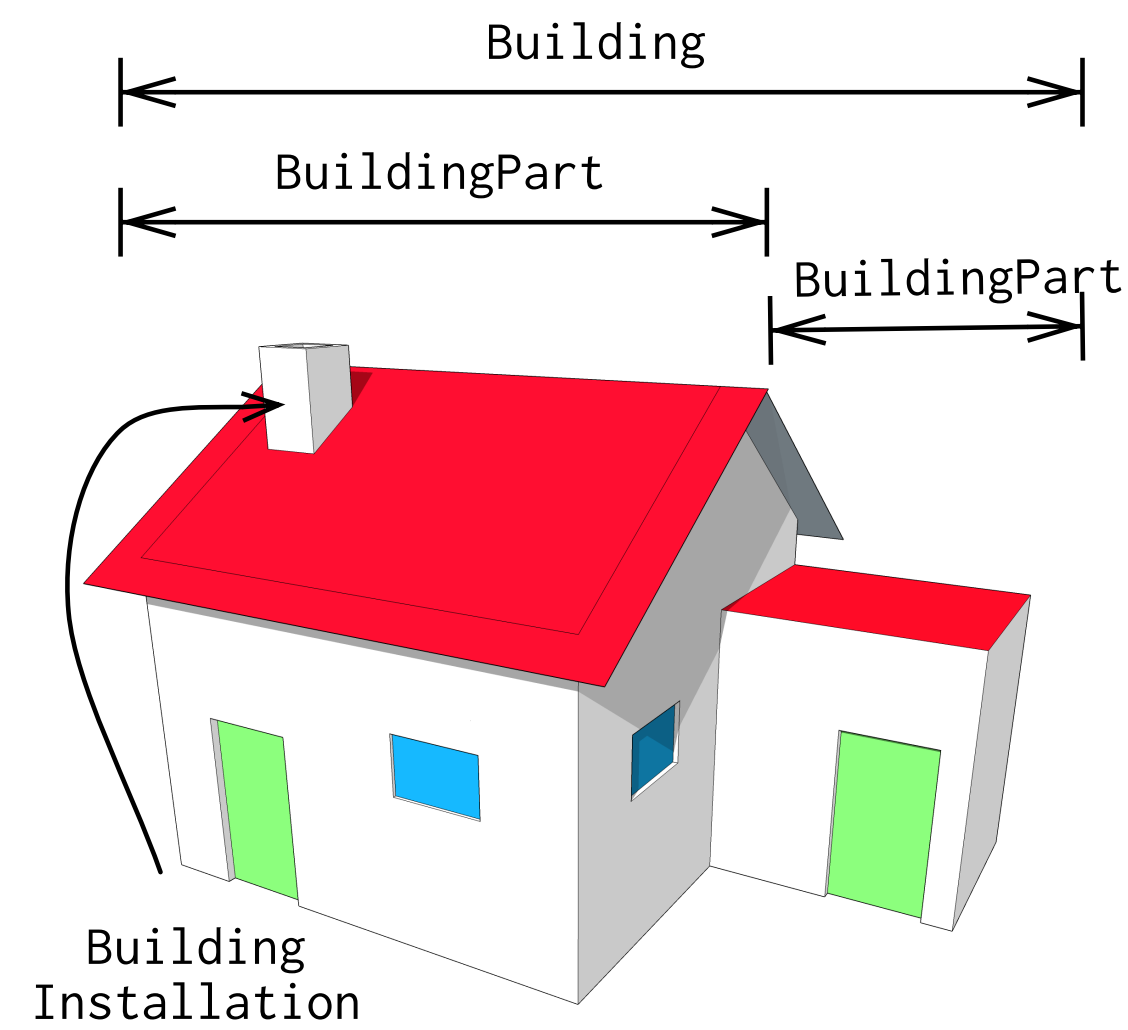
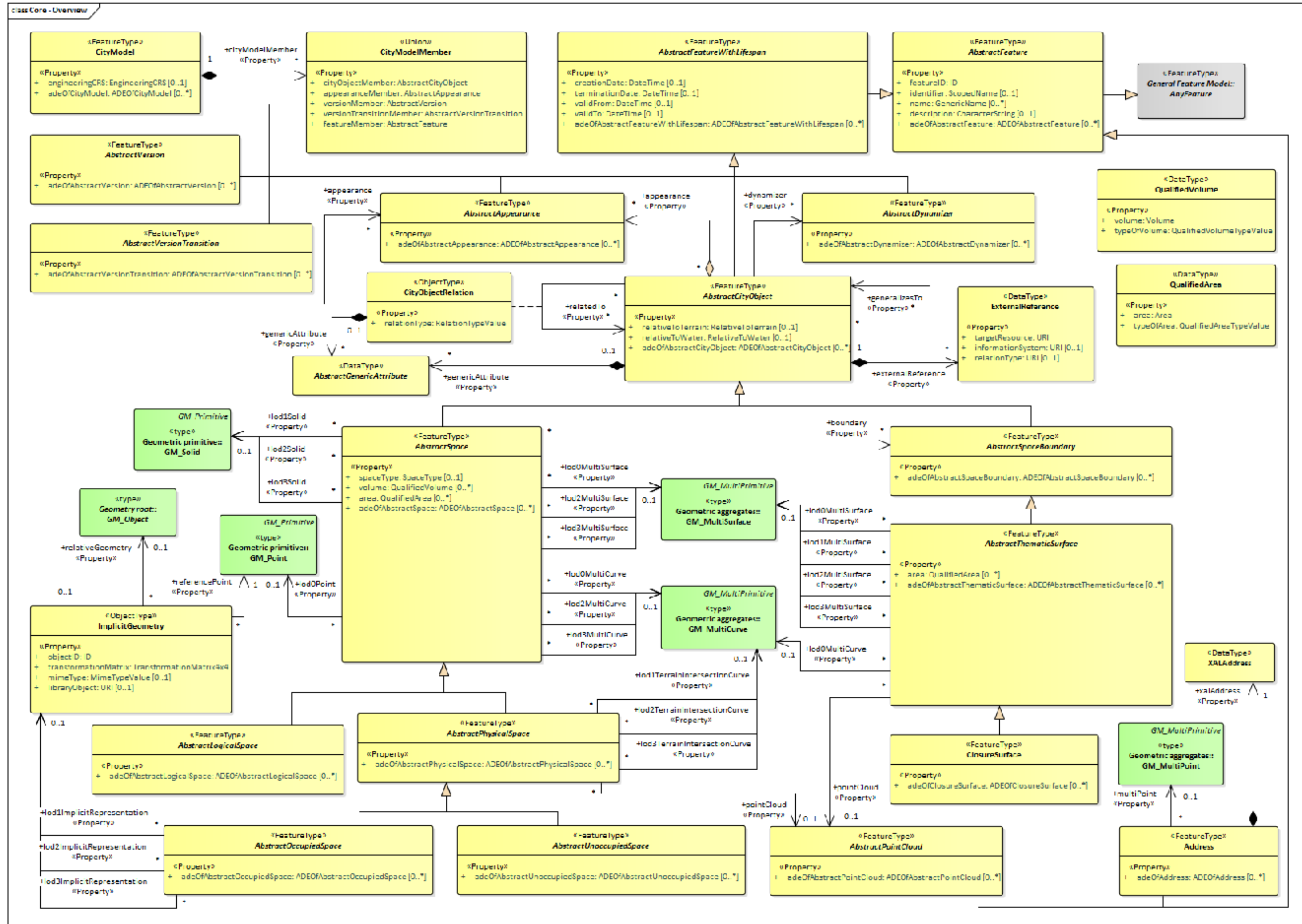
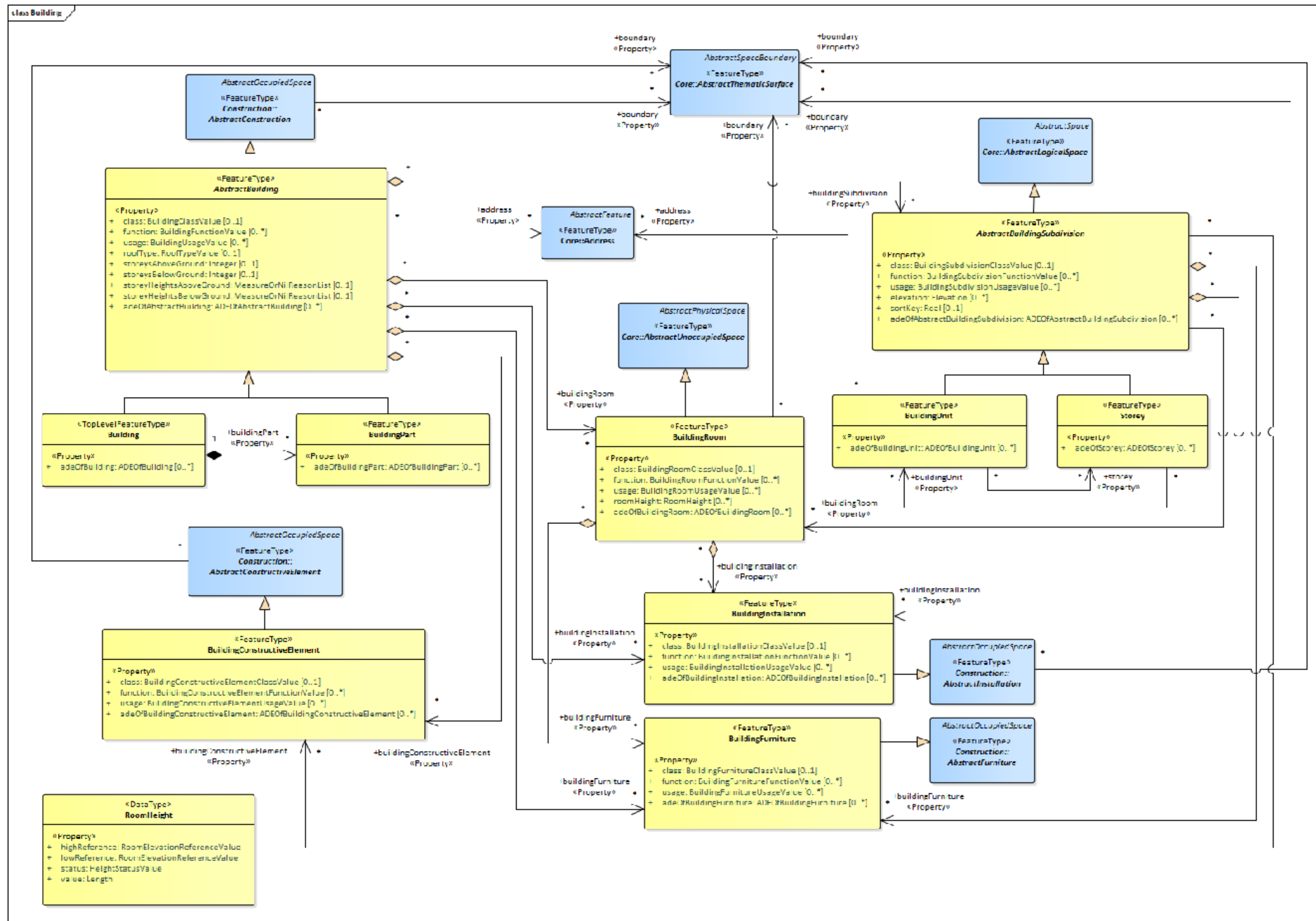


Figure 2: A building is semantically decomposed into different objects, and each objects is defined with geometry.

CityGML == a data model == UML models



CityGML == a data model == UML models



Levels of detail (LoDs)

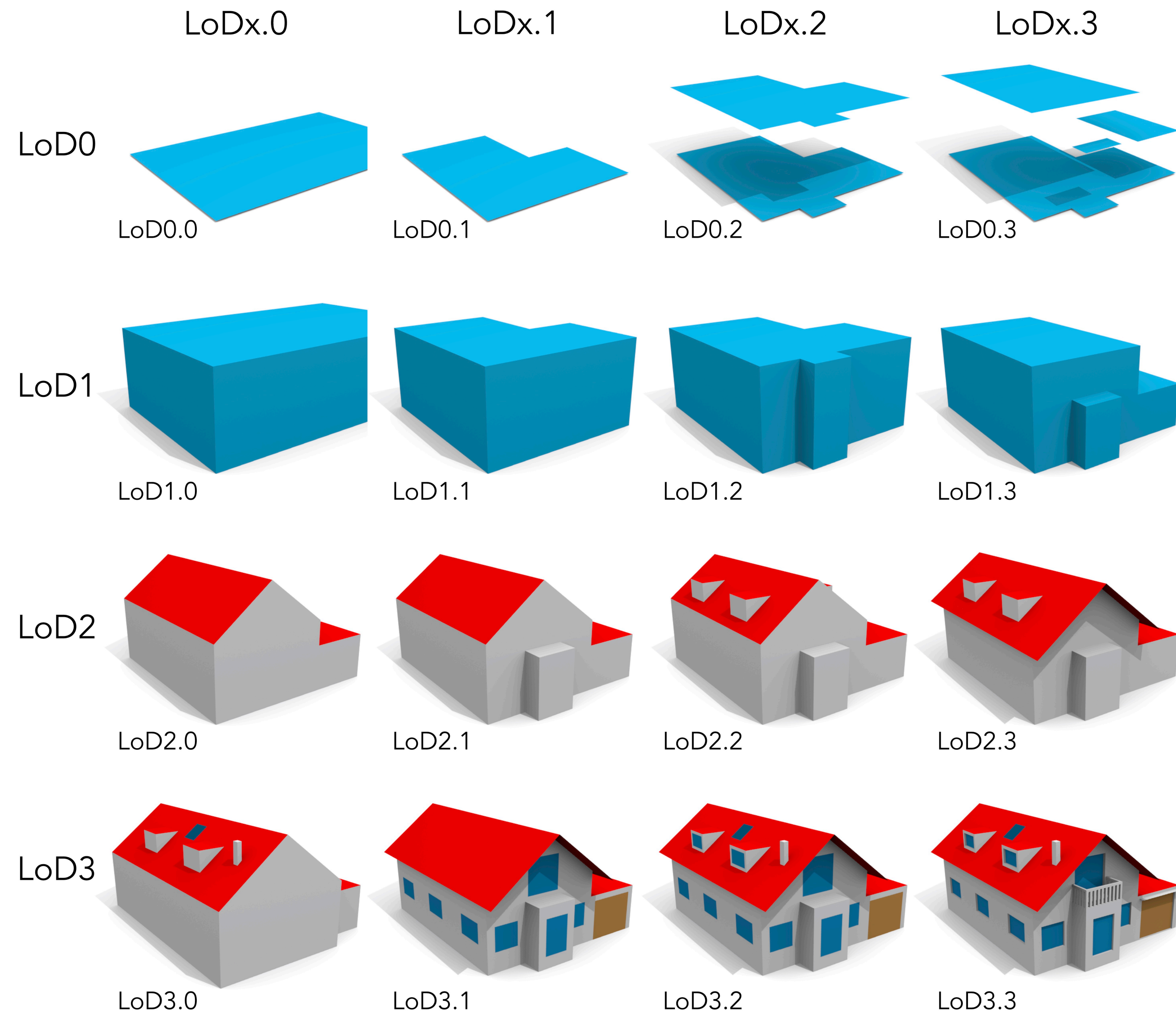


Figure 7: The improved LoDs for buildings; they are generally referred to as the *TU Delft LoDs*.

XML-encoded CityGML files

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <CityModel xmlns:xlink="http://www.w3.org/1999/xlink"
3   xmlns:gml="http://www.opengis.net/gml"
4   xmlns="http://www.opengis.net/citygml/2.0"
5   xmlns:bldg="http://www.opengis.net/citygml/building/2.0"
6   xsi:schemaLocation="http://www.opengis.net/citygml/2.0">
7   <cityObjectMember>
8     <bldg:Building gml:id="9a06451677c7">
9       <bldg:function>1070</bldg:function>
10      <bldg:lod1Solid>
11        <gml:Solid>
12          <gml:exterior>
13            <gml:CompositeSurface>
14              <gml:surfaceMember>
15                <gml:Polygon>
16                  <gml:exterior>
17                    <gml:LinearRing>
18                      <gml:pos>0.0 0.0 0.0</gml:pos>
19                      <gml:pos>0.0 1.0 0.0</gml:pos>
20                      <gml:pos>1.0 1.0 0.0</gml:pos>
21                      <gml:pos>1.0 0.0 0.0</gml:pos>
22                      <gml:pos>0.0 0.0 0.0</gml:pos>
23                    </gml:LinearRing>
24                  </gml:exterior>
25                </gml:Polygon>
26              </gml:surfaceMember>
27            ...
28          </bldg:Building>
29          <bldg:Building gml:id="jdhd76sa">
30            ...
31          </bldg:Building>
32        </cityObjectMember>
33      </CityModel>
```

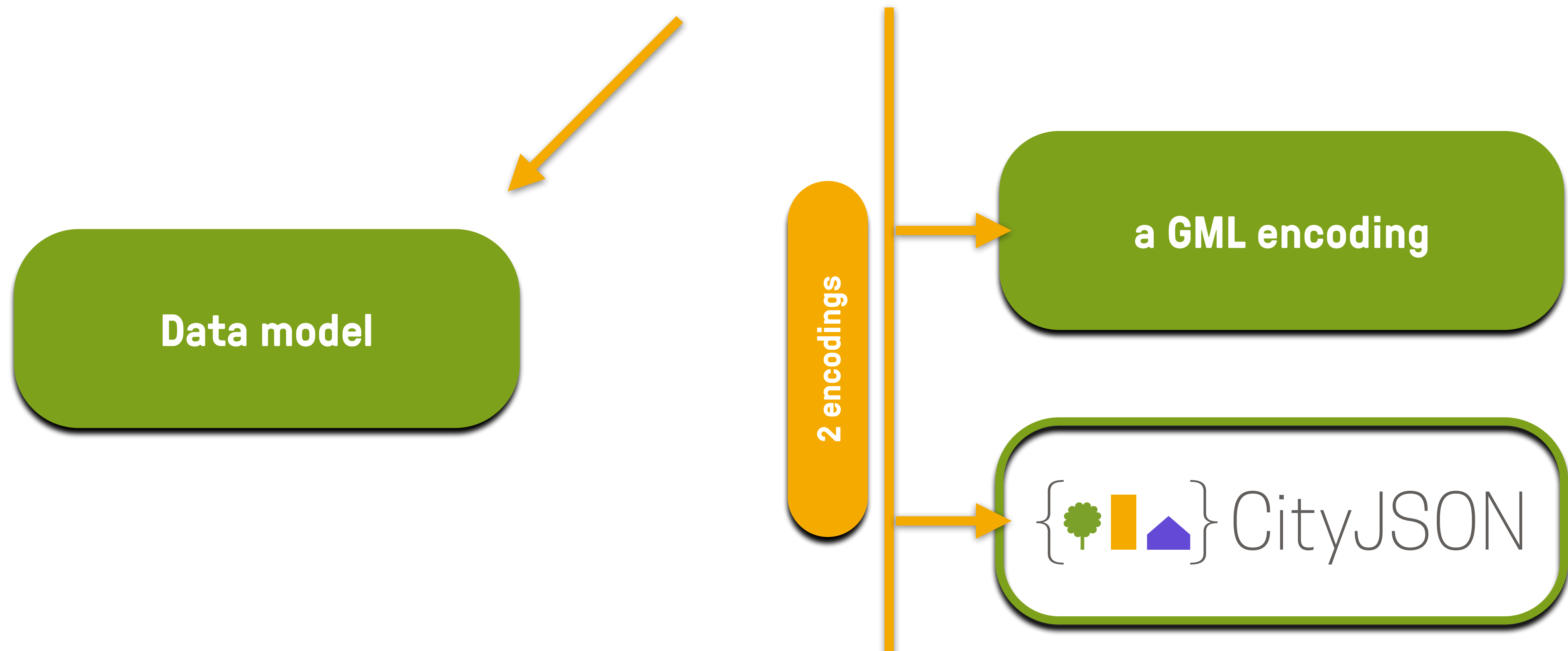
Figure 1.11: Part of a CityGML file containing 2 buildings.

CityGML files are very complex

- files are deeply nested, and large
- many “points of entry”
- many diff ways to do one thing

- ➔ few software packages use CityGML
- ➔ no parsers in JavaScript
- ➔ I personally get 😞 each time I get a new file





Why an alternative encoding? Read this.

Ledoux et al. *Open Geospatial Data, Software and Standards*
<https://doi.org/10.1186/s40965-019-0064-0>

(2019) 4:4

Open Geospatial Data,
Software and Standards

ORIGINAL ARTICLE

Open Access

CityJSON: a compact and easy-to-use encoding of the CityGML data model



Hugo Ledoux^{*} , Ken Arroyo Ohori, Kavisha Kumar, Balázs Dukai, Anna Labetski and Stelios Vitalis

Abstract

The international standard CityGML is both a data model and an exchange format to store digital 3D models of cities. While the data model is used by several cities, companies, and governments, in this paper we argue that its XML-based exchange format has several drawbacks. These drawbacks mean that it is difficult for developers to implement parsers for CityGML, and that practitioners have, as a consequence, to convert their data to other formats if they want to exchange them with others. We present CityJSON, a new JSON-based exchange format for the CityGML data model (version 2.0.0). CityJSON was designed with programmers in mind, so that software and APIs supporting it can be quickly built. It was also designed to be compact (a compression factor of around six with real-world datasets), and to be friendly for web and mobile development. We argue that it is considerably easier to use than the CityGML format, both for reading and for creating datasets. We discuss in this paper the main features of CityJSON, briefly

Home | CityJSON

https://www.cityjson.org

CityJSON

Datasets

Extensions

Software

Schemas

Specifications

Experimental

Help for developers

Tutorials

Search CityJSON

{

CityJSON

A JSON-based encoding for 3D city models

Specs latest version: v1.1.3

Getting started

Web-viewer

Validator

Latest news

25 NOV 2022 [CityJSON v1.1.3 released](#)

Same information as CityGML, but in JSON format

```
{
  "type": "CityJSON",
  "version": "2.0",
  "metadata": {
    "referenceSystem": "https://www.opengis.net/def/crs/EPSG/0/7415"
  },
  "transform": {...}
  "CityObjects": {
    "id-1": {
      "type": "Building",
      "attributes": {
        "measuredHeight": 22.3,
        "roofType": "gable",
        "owner": "Elvis Presley"
      },
      "geometry": [
        {
          "type": "MultiSurface",
          "boundaries": [
            [[0, 3, 2, 1]], [[4, 5, 6, 7]], [[0, 1, 5, 4]]
          ]
        }
      ]
    }
  ],
  "vertices": [
    [231, 23212, 110],
    [1111, 3211, 120],
    ...
  ],
  "appearance": {
    "materials": [],
    "textures": [],
    "vertices-texture": []
  }
}
```

CityJSON

CityGML-XML files are very complex

- files are deeply nested, and large
- many “points of entry”
- many diff ways to do one thing

- ➔ few software packages use CityGML
- ➔ no parsers in JavaScript
- ➔ I personally get 😞 each time I get a new file



MSc Geomatics students when trying to parse a CityGML file in Python



version 1.1 = current version

2.0 (since September 2023)

-  official community standard
- compliant with CityGML v3.0
- subset of CityGML (~90% of features)
- software for full conversion CityGML <-> CityJSON
- several software support CityJSON

1st-level City Objects

Bridge
Building
CityFurniture
CityObjectGroup
GenericCityObject
LandUse
OtherConstruction
PlantCover
SolitaryVegetationObject
TINRelief
TransportationSquare
Railway
Road
Tunnel
WaterBody
Waterway
+Extension

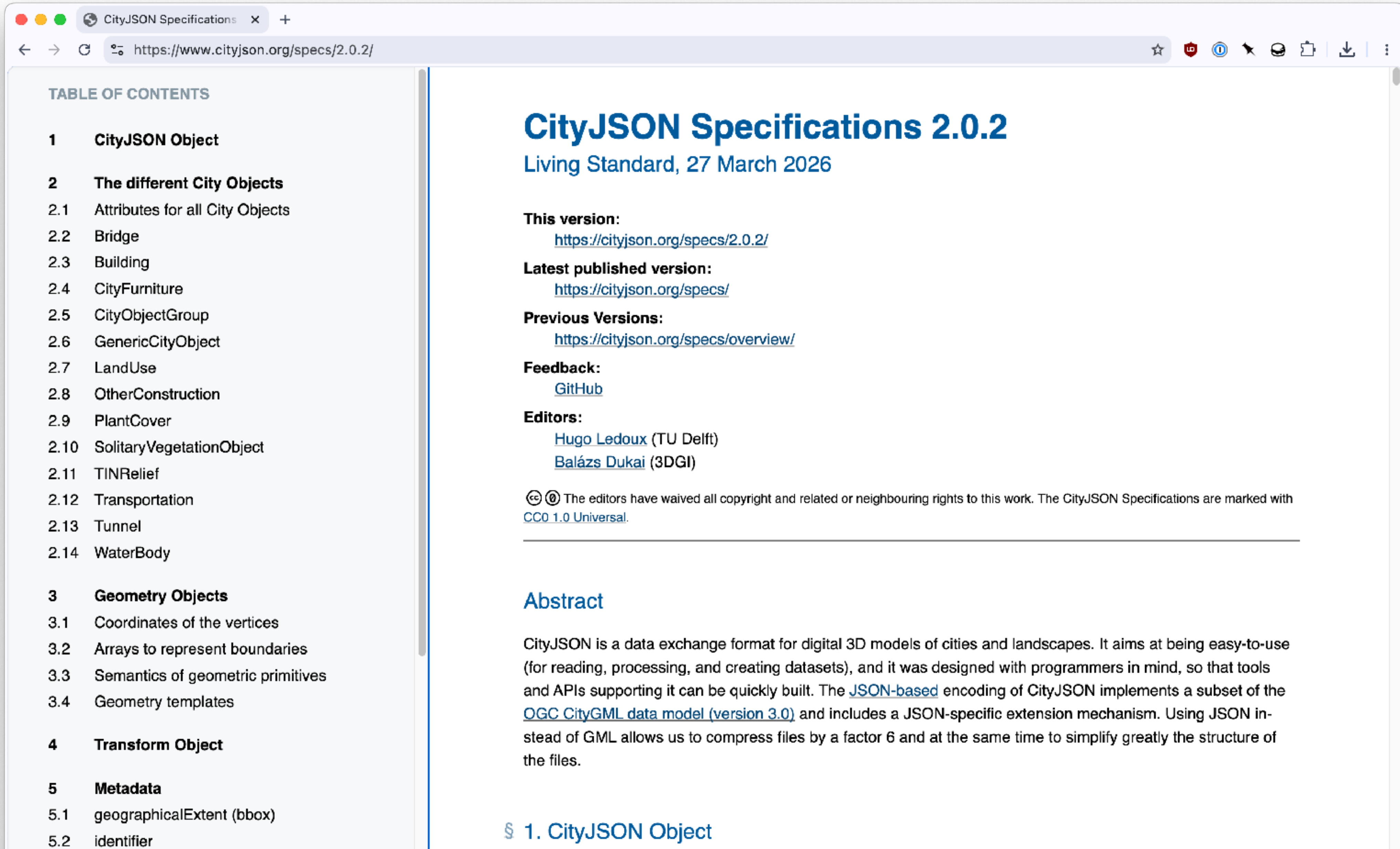
2nd-level City Objects

BridgePart
BridgeInstallation
BridgeConstructiveElement
BridgeRoom
BridgeFurniture

BuildingPart
BuildingInstallation
BuildingConstructiveElement
BuildingFurniture
BuildingStorey
BuildingRoom
BuildingUnit

TunnelPart
TunnelInstallation
TunnelConstructiveElement
TunnelHollowSpace
TunnelFurniture

Specifications give all the gory details



CityJSON Specifications 2.0.2

Living Standard, 27 March 2026

This version:
<https://cityjson.org/specs/2.0.2/>

Latest published version:
<https://cityjson.org/specs/>

Previous Versions:
<https://cityjson.org/specs/overview/>

Feedback:
[GitHub](#)

Editors:
[Hugo Ledoux](#) (TU Delft)
[Balázs Dukai](#) (3DGI)

© The editors have waived all copyright and related or neighbouring rights to this work. The CityJSON Specifications are marked with [CC0 1.0 Universal](#).

Abstract

CityJSON is a data exchange format for digital 3D models of cities and landscapes. It aims at being easy-to-use (for reading, processing, and creating datasets), and it was designed with programmers in mind, so that tools and APIs supporting it can be quickly built. The [JSON-based](#) encoding of CityJSON implements a subset of the [OGC CityGML data model \(version 3.0\)](#) and includes a JSON-specific extension mechanism. Using JSON instead of GML allows us to compress files by a factor 6 and at the same time to simplify greatly the structure of the files.

§ 1. CityJSON Object

1	CityJSON Object
2	The different City Objects
2.1	Attributes for all City Objects
2.2	Bridge
2.3	Building
2.4	CityFurniture
2.5	CityObjectGroup
2.6	GenericCityObject
2.7	LandUse
2.8	OtherConstruction
2.9	PlantCover
2.10	SolitaryVegetationObject
2.11	TINRelief
2.12	Transportation
2.13	Tunnel
2.14	WaterBody
3	Geometry Objects
3.1	Coordinates of the vertices
3.2	Arrays to represent boundaries
3.3	Semantics of geometric primitives
3.4	Geometry templates
4	Transform Object
5	Metadata
5.1	geographicalExtent (bbox)
5.2	identifier

A CityJSON file

```
{
  "type": "CityJSON",
  "version": "2.0",
  "metadata": {
    "referenceSystem": "https://www.opengis.net/def/crs/EPSG/0/7415"
  },
  "transform": {...}
  "CityObjects": {
    "id-1": {
      "type": "Building",
      "attributes": {
        "measuredHeight": 22.3,
        "roofType": "gable",
        "owner": "Elvis Presley"
      },
      "geometry": [
        {
          "type": "MultiSurface",
          "boundaries": [
            [[0, 3, 2, 1]], [[4, 5, 6, 7]], [[0, 1, 5, 4]]
          ]
        }
      ]
    }
  ],
  "vertices": [
    [231, 23212, 110],
    [1111, 3211, 120],
    ...
  ],
  "appearance": {
    "materials": [],
    "textures": [],
    "vertices-texture": []
  }
}
```

version CityJSON

A CityJSON file

```
{
  "type": "CityJSON",
  "version": "2.0",
  "metadata": {
    "referenceSystem": "https://www.opengis.net/def/crs/EPSG/0/7415"
  },
  "transform": {...}
  "CityObjects": {
    "id-1": {
      "type": "Building",
      "attributes": {
        "measuredHeight": 22.3,
        "roofType": "gable",
        "owner": "Elvis Presley"
      },
      "geometry": [
        {
          "type": "MultiSurface",
          "boundaries": [
            [[0, 3, 2, 1]], [[4, 5, 6, 7]], [[0, 1, 5, 4]]
          ]
        }
      ]
    }
  },
  "vertices": [
    [231, 23212, 110],
    [1111, 3211, 120],
    ...
  ],
  "appearance": {
    "materials": [],
    "textures": [],
    "vertices-texture": []
  }
}
```

metadata possible, eg:

- CRS
- Point of contact
- bbox of dataset

A CityJSON file

```
{
  "type": "CityJSON",
  "version": "2.0",
  "metadata": {
    "referenceSystem": "https://www.opengis.net/def/crs/EPSG/0/7415"
  },
  "transform": [...],
  "CityObjects": [
    "id-1": {
      "type": "Building",
      "attributes": {
        "measuredHeight": 22.3,
        "roofType": "gable",
        "owner": "Elvis Presley"
      },
      "geometry": [
        {
          "type": "MultiSurface",
          "boundaries": [
            [[0, 3, 2, 1]], [[4, 5, 6, 7]], [[0, 1, 5, 4]]
          ]
        }
      ]
    }
  ],
  "vertices": [
    [231, 23212, 110],
    [1111, 3211, 120],
    ...
  ],
  "appearance": {
    "materials": [],
    "textures": [],
    "vertices-texture": []
  }
}
```

All City Objects listed here, *indexed* by their ID

Each have geometries + attributes

A CityJSON file

```
{
  "type": "CityJSON",
  "version": "2.0",
  "metadata": {
    "referenceSystem": "https://www.opengis.net/def/crs/EPSG/0/7415"
  },
  "transform": {...}
  "CityObjects": {
    "id-1": {
      "type": "Building",
      "attributes": {
        "measuredHeight": 22.3,
        "roofType": "gable",
        "owner": "Elvis Presley"
      },
      "geometry": [
        {
          "type": "MultiSurface",
          "boundaries": [
            [[0, 3, 2, 1]], [[4, 5, 6, 7]], [[0, 1, 5, 4]]
          ]
        }
      ]
    },
    "vertices": [
      [231, 23212, 110],
      [1111, 3211, 120],
      ...
    ],
    "appearance": {
      "materials": [],
      "textures": [],
      "vertices-texture": []
    }
  }
}
```



Geometry is ID of the vertex, global list

compression + more “topology”

0-indexed

A CityJSON file

```
{
  "type": "CityJSON",
  "version": "2.0",
  "metadata": {
    "referenceSystem": "https://www.opengis.net/def/crs/EPSG/0/7415"
  },
  "transform": {...}
  "CityObjects": {
    "id-1": {
      "type": "Building",
      "attributes": {
        "measuredHeight": 22.3,
        "roofType": "gable",
        "owner": "Elvis Presley"
      },
      "geometry": [
        {
          "type": "MultiSurface",
          "boundaries": [
            [[0, 3, 2, 1]], [[4, 5, 6, 7]], [[0, 1, 5, 4]]
          ]
        }
      ]
    }
  ],
  "vertices": [
    [231, 23212, 110],
    [1111, 3211, 120],
    ...
  ],
  "appearance": {
    "materials": [],
    "textures": [],
    "vertices-texture": []
  }
}
```

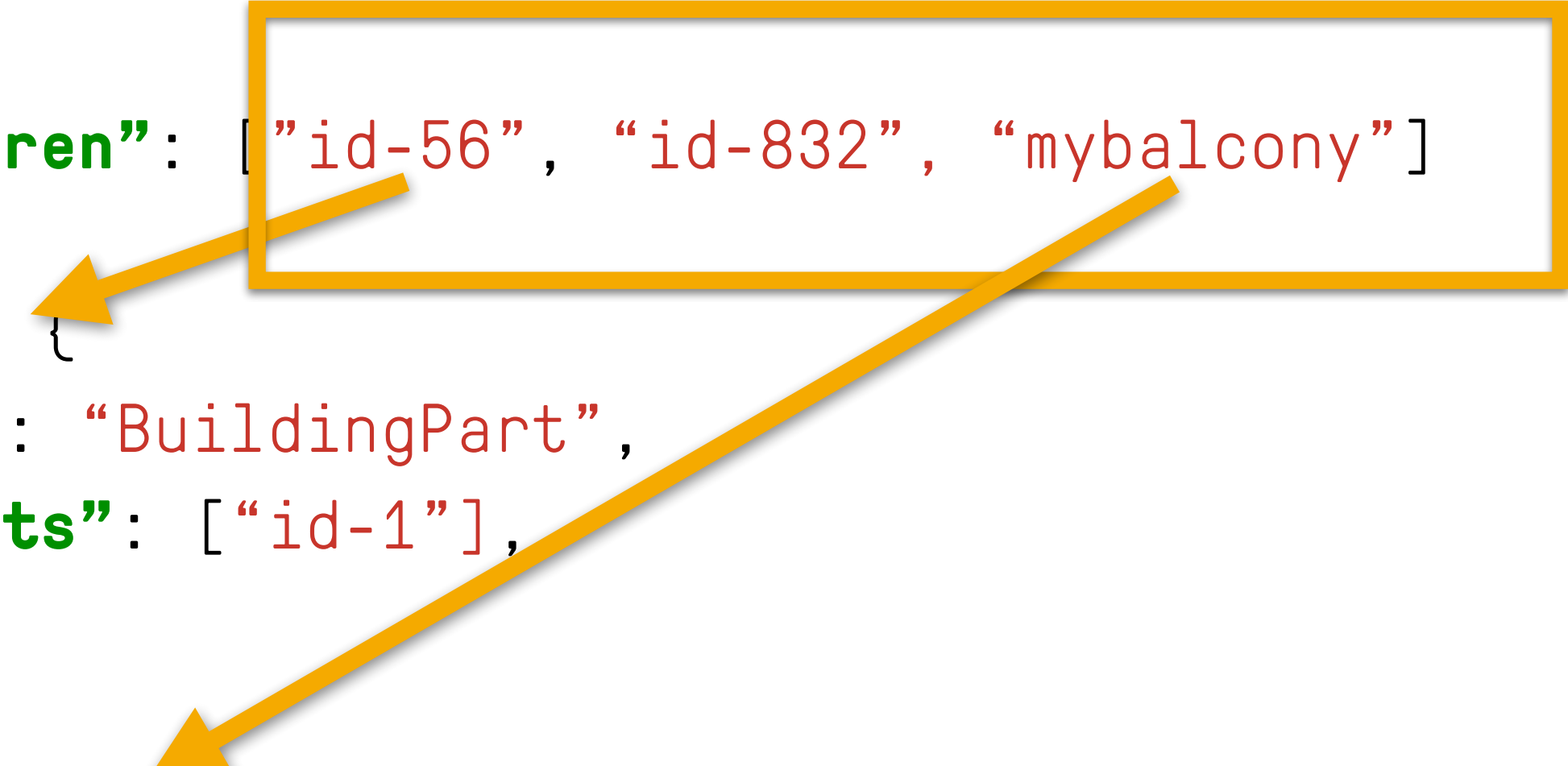
materials + textures possible

BuildingParts: links between City Objects (file is flattened)

```
"CityObjects": {  
  "id-1": {  
    "type": "Building",  
    "attributes": {  
      "roofType": "gable"  
    },  
    "children": ["id-56", "id-832", "mybalcony"]  
  },  
  "id-56": {  
    "type": "BuildingPart",  
    "parents": ["id-1"],  
    ...  
  },  
  "mybalcony": {  
    "type": "BuildingInstallation",  
    "parent": ["id-1"],  
    ...  
  }  
}
```


BuildingParts: links between City Objects (file is flattened)

```
"CityObjects": {  
  "id-1": {  
    "type": "Building",  
    "attributes": {  
      "roofType": "gable"  
    },  
    "children": ["id-56", "id-832", "mybalcony"]  
  },  
  "id-56": {  
    "type": "BuildingPart",  
    "parents": ["id-1"],  
    ...  
  },  
  "mybalcony": {  
    "type": "BuildingInstallation",  
    "parents": ["id-1"],  
    ...  
  }  
}
```



goal == a flat schema

CityJSON software

web-viewer: ninja.cityjson.org

The screenshot shows the CityJSON Ninja web viewer interface. The browser address bar displays `https://ninja.cityjson.org/#`. The interface includes a sidebar with the 'ninja' logo and 'Objects' section, a main panel with a search bar and a list of 2498 total city objects, and a detailed view of a selected building.

City Objects 2498 total

Search for IDs, object type or attributes...

[Download](#) [Close](#)

GUID_F7FB3F3C-1D1C-48FD-B106-B6A35C789DB5 [LoD2](#)

> GUID_54E79928-D82D-4FC8-B2FD-5BA0CB4E5535

GUID_2DA36D01-8199-466D-A8E6-C2672A058756 [LoD2](#)

> GUID_46A81A33-C715-4A2A-8AAF-5398EA1191F0

GUID_5B6298F2-C103-4F63-981F-96F3194C00E3 [LoD2](#)

GUID_DCCDE7FB-577E-4E32-9110-9E8BB8EB1080 [LoD2](#)

> GUID_88D03B4B-0F50-4B53-AB5D-B4D738B44768

GUID_FBC35DA1-A388-4EA3-A4EA-68703A790603 [LoD2](#)

> GUID_8F89178C-221D-405E-A214-E099149432E3

> GUID_957B7FE8-F71F-46B2-9956-DCAB6293A2AD

> GUID_E30876AC-FF11-44FF-9D19-31AF152C50E1

> GUID_FECF6A22-5628-4577-BD8E-714701F84F8A

GUID_D7E9C2DC-AD1D-4F3F-

Building [Edit](#) [Close](#)

GUID_FBC35DA1-A388-4EA3-A4EA-68703A790603

5 Attributes [1 Geometries](#)

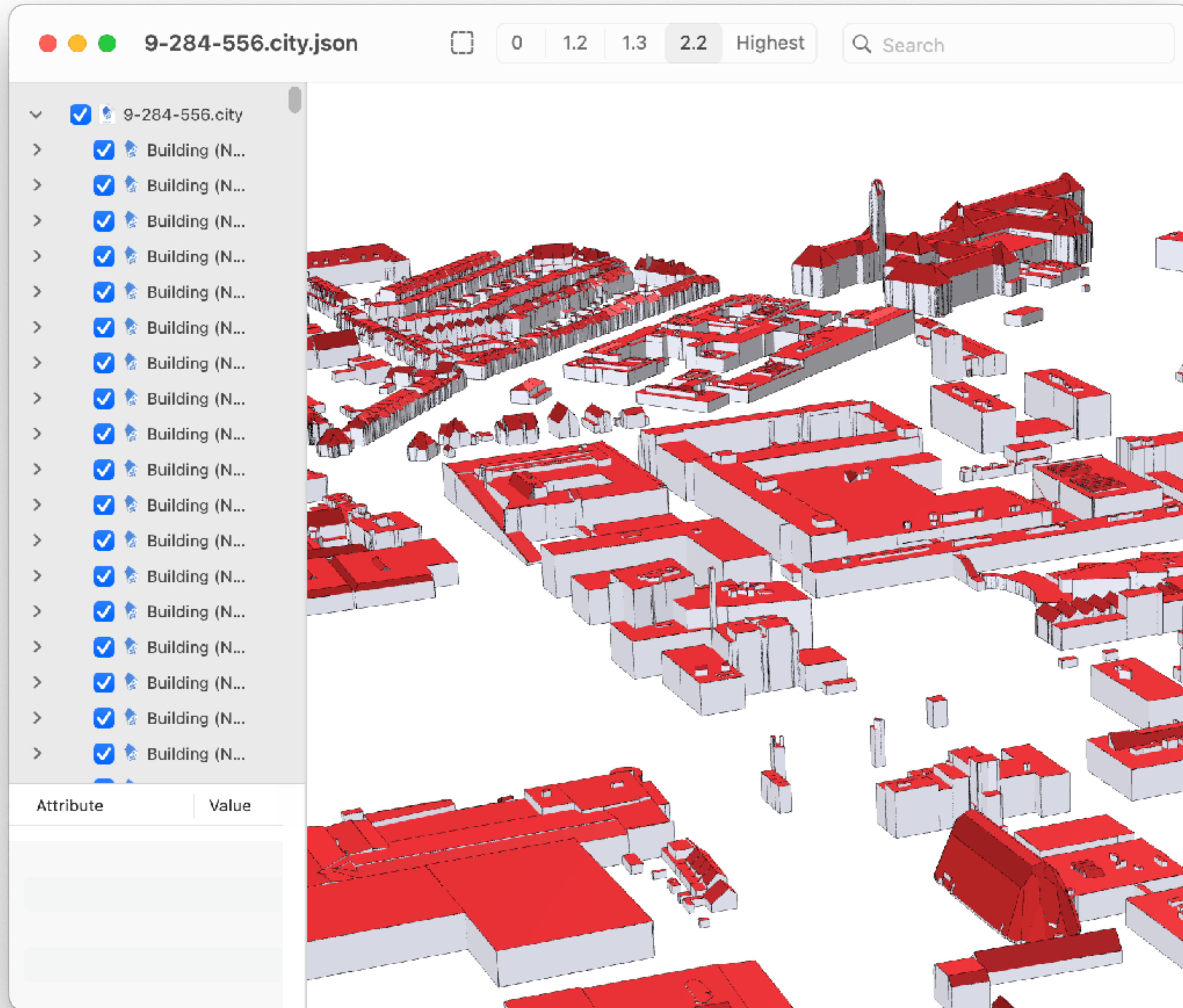
roofType	1000
RelativeEavesHeight	9.833
RelativeRidgeHeight	9.833
AbsoluteEavesHeight	16.181
AbsoluteRidgeHeight	16.181

The bottom of the interface shows a 3D visualization of a city model with blue buildings on a yellow ground plane.

Nice fact: programmed mostly by 2 Geomatics students, as projects for GEO5010

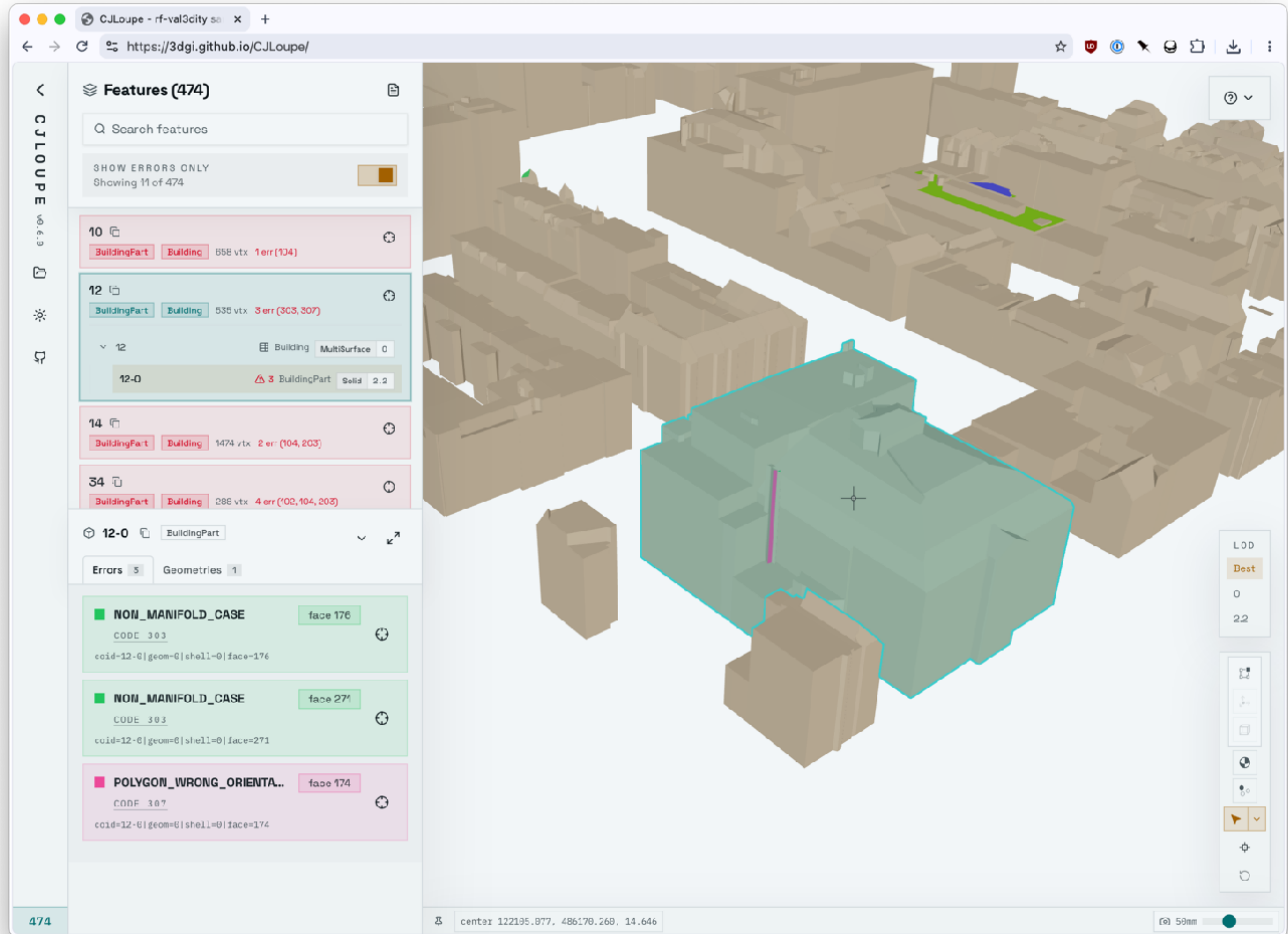
You can do the same (or similar!)

macOS viewer: azul (from Ken)



It's in the App Store

CJLoupe



Validation of the syntax of a file: cjval

A GE05010 project from a student also kick-started this!

The screenshot shows the CityJSON schema validator interface. At the top, there's a logo and a button to "drop a CityJSON file or click to browse". Below this, a purple box contains the title "CityJSON schema validator" and notes that it uses "cjval v0.9.0" and supports CityJSON v2.0, v1.1, and v1.0. It also states that CityJSONSeq and Extensions are supported, validation is done locally, and geometries are not validated. The file being validated is "9-284-556.city.json", version "CityJSON v2.0 (schemas used: v2.0.2)", with no extensions. A yellow banner indicates "The file is valid but has warnings". Below this, there are two columns: "ERRORS" and "WARNINGS". The "ERRORS" column lists several categories: JSON syntax, CityJSON schemas, Extension schemas, parents/children consistency, wrong vertex index, semantics arrays, and materials. The "WARNINGS" column lists "duplicate vertices" (with a list of duplicated vertex coordinates) and "extra root properties".

CityJSON schema validator
(uses [cjval](#) v0.9.0)

Supports CityJSON v2.0, v1.1, and v1.0
CityJSONSeq and Extensions also supported
Validation is done locally, files are never uploaded
Geometries are **not** validated (details)

File: 9-284-556.city.json
Version: CityJSON v2.0 (schemas used: v2.0.2)
Extensions: none

● The file is valid but has warnings

ERRORS

- JSON syntax
- CityJSON schemas
- Extension schemas
- parents/children consistency
- wrong vertex index
- semantics arrays
- materials

WARNINGS

- duplicate vertices**
Vertex [-279754, -76289, -41137] duplicated
Vertex [-289502, -71979, -41137] duplicated
Vertex [-279754, -76289, -34825] duplicated
Vertex [-289502, -71979, -34814] duplicated
Vertex [-272817, -23685, -34838] duplicated
Vertex [-251750, -69807, -32082] duplicated
Vertex [-380338, -49483, -41137] duplicated
Vertex [-386394, -45303, -41137] duplicated
Vertex [-297500, -30518, -41137] duplicated
Vertex [-285061, -25263, -41137] duplicated
... and 1857 more
- extra root properties
- unused vertices

```
{
  "type": "CityJSON",
  "version": "2.0",
  "metadata": {
    "referenceSystem": "https://www.opengis.net/def/crs/EPSG/0/7415"
  },
  "transform": {...}
  "CityObjects": {
    "id-1": {
      "type": "Buildnig",
      "attributes": {
        "measuredHeight": 22.3,
        "roofType": "gable",
        "owner": "Elvis Presley"
      },
      "geometry": [
        {
          "type": "MultiSurface",
          "boundaries": [
            [[0, 3, 2, 1]], [[4, 5, 6, 7]], [[0, 1, 5, 4]]
          ]
        }
      ]
    },
    "vertices": [
      [231, 23212, 110.223],
      [1111, 3211, 120],
      ...
    ],
    "appearance": {
      "materials": [],
      "textures": [],
      "vertices-texture": []
    }
  }
}
```


Python parser is simple



```
import json

fin = open('mycity.json')
cm = json.loads(fin.read())

print "There are", len(cm['City0bjects']), "City0bjects"

# list all ids
for id in cm['City0bjects']:
    print "\t", id
```

cjio (CityJSON/io) => pip install cjio

```
2. bash
Hugos-MacBook-Pro:rotterdam hugo$ cjio
Usage: cjio [OPTIONS] INPUT COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...

Process and manipulate a CityJSON file, and allow different outputs. The
different operators can be chained to perform several processing in one
step, the CityJSON model goes through the different operators.

To get help on specific command, eg for 'validate':

    cjio validate --help

Usage examples:

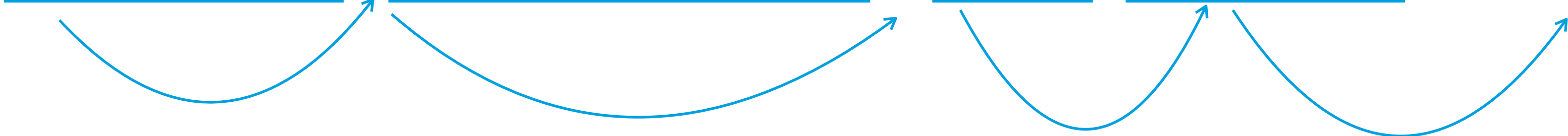
    cjio example.json validate
    cjio example.json remove_textures info
    cjio example.json subset --id house12 remove_materials save out.json

Options:
  --version          Show the version and exit.
  --off              Load an OFF file and convert it to one CityJSON
                     GenericCityObject.
  --ignore_duplicate_keys Load a CityJSON file even if some City Objects have
                     the same IDs (technically invalid file)
  --help             Show this message and exit.

Commands:
  compress          Compress a CityJSON file, ie stores its...
  decompress        Decompress a CityJSON file, ie remove the...
  info              Output info in simple JSON.
  merge             Merge the current CityJSON with others.
  remove_duplicate_vertices Remove duplicate vertices a CityJSON file.
  remove_materials  Remove all materials from a CityJSON file.
  remove_orphan_vertices Remove orphan vertices a CityJSON file.
  remove_textures   Remove all textures from a CityJSON file.
  save              Save the CityJSON to a file.
  subset            Create a subset of a CityJSON file.
  update_bbox       Update the bbox of a CityJSON file.
  update_crs        Update the CRS with a new value.
  validate          Validate the CityJSON file: (1) against its...
```

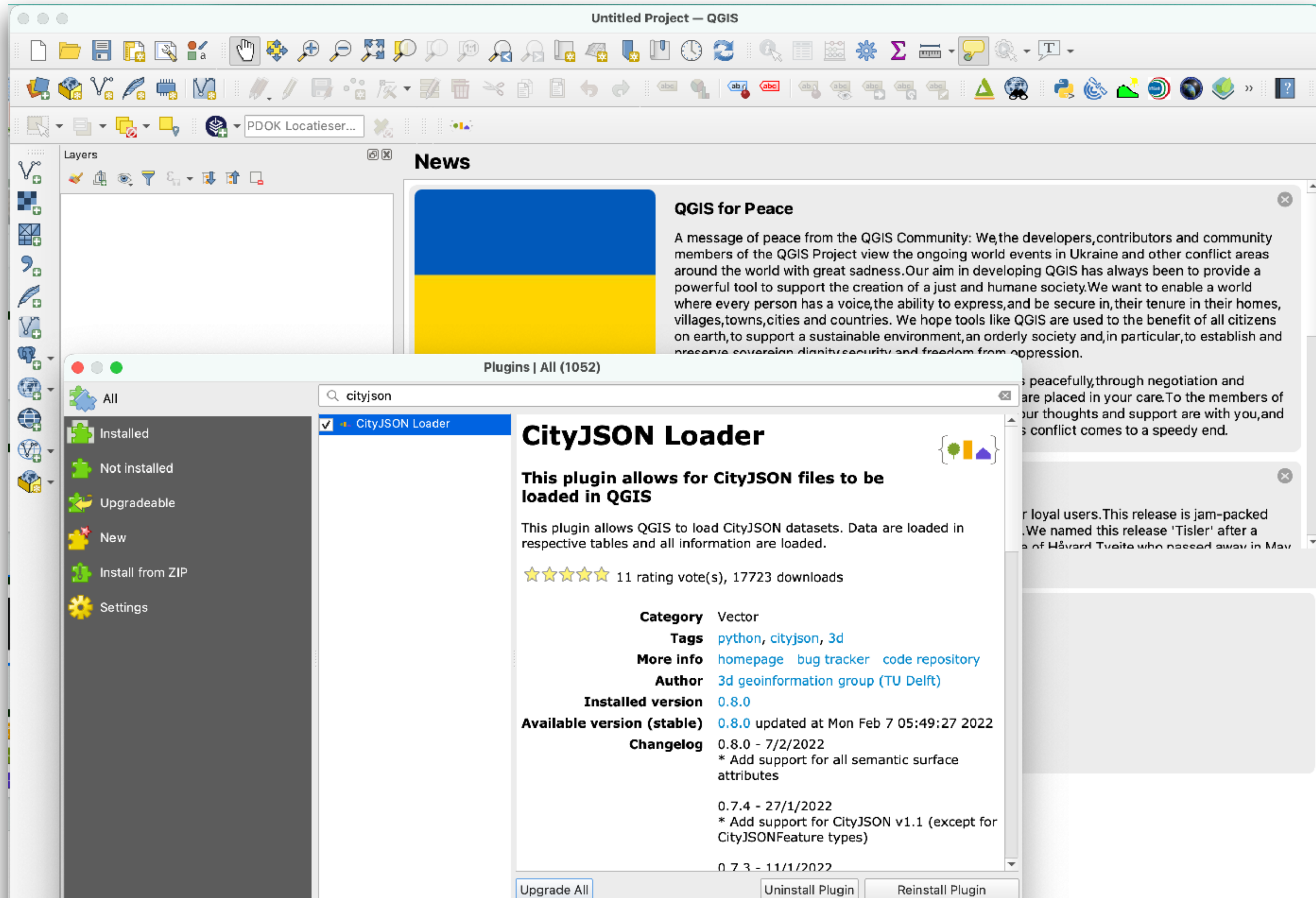


```
$ cjio myfile.json crs_assign 7415 subset --cotype Building lod_filter 2.2 save out.json
```

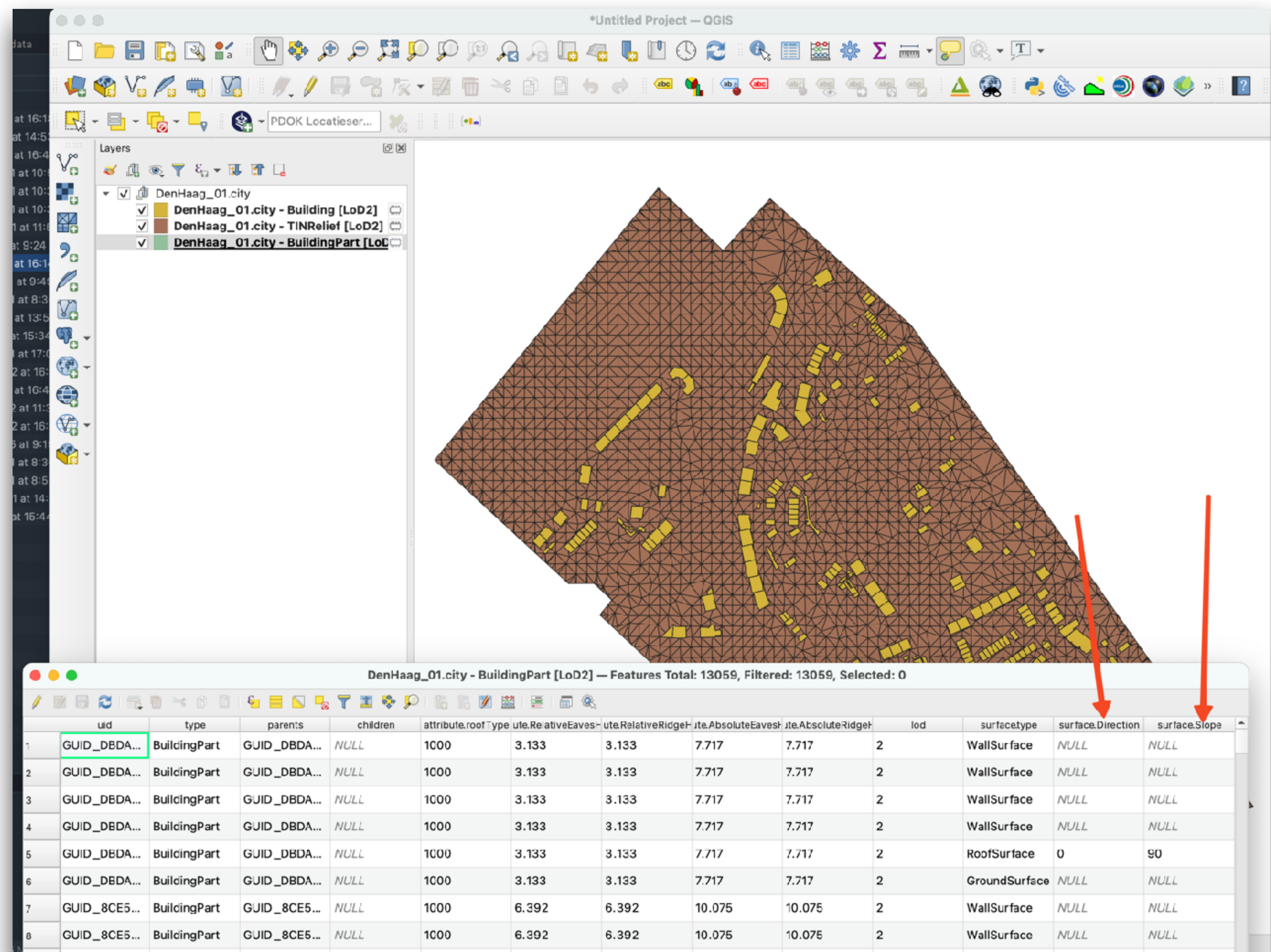


**No need to save temp files between
operators: pipeline is used**

QGIS plugin



QGIS plugin



citygml4j/citygml4j: The Open Source Java API for CityGML

521 commits 1 branch 25 releases 1 contributor Apache-2.0

Branch: master New pull request Create new file Upload files Find file Clone or download

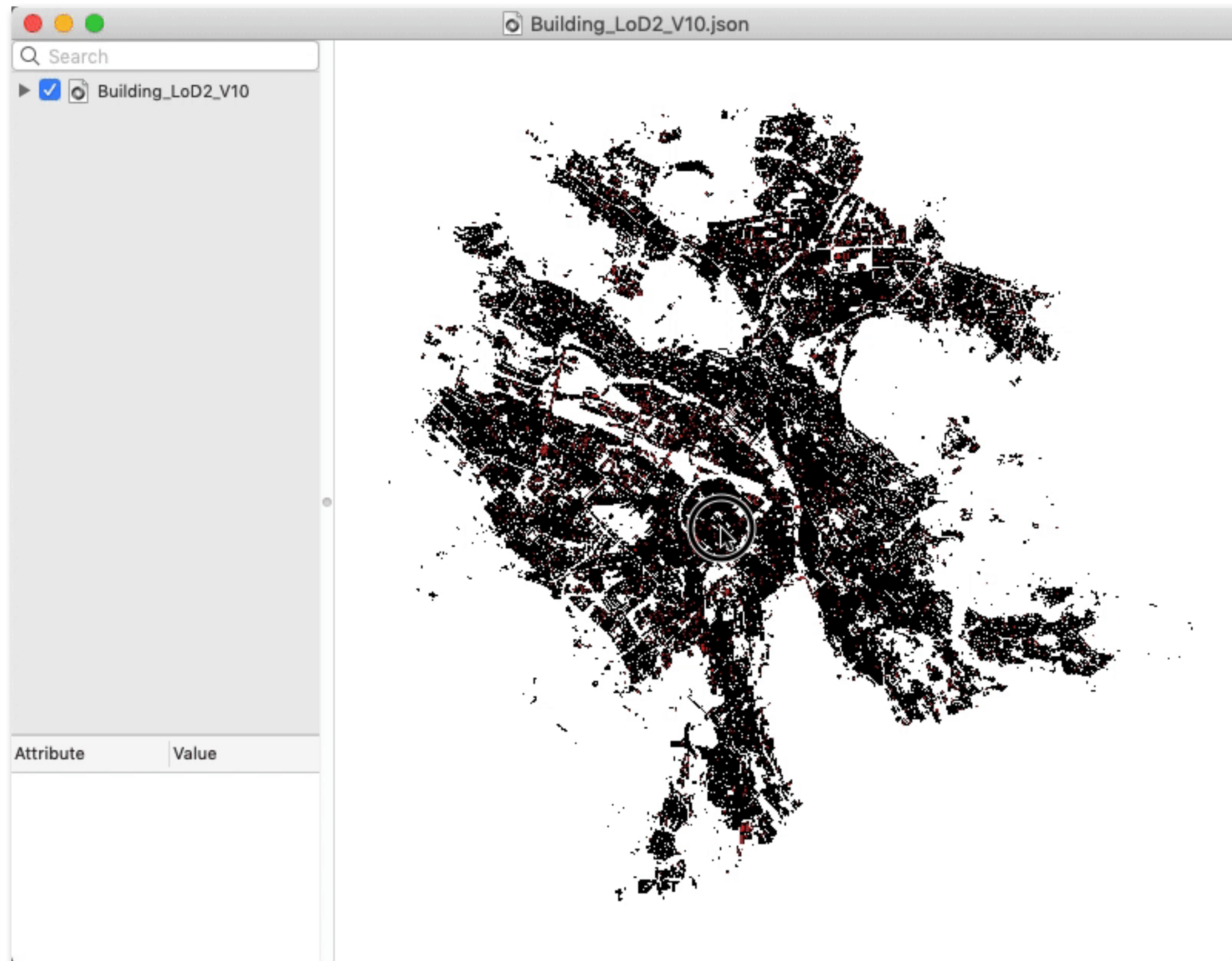
citygml4j	added generated JAXB classes	3 months ago
src-gen/main/java	added generated JAXB classes	3 months ago
src/main	removed unnecessary properties from CityJSON input and output factories	2 months ago
.gitignore	minor change	3 months ago
LICENSE	changes license to Apache License, Version 2.0	2 years ago
README.md	Update README.md	2 months ago
build.gradle	updated gradle	a month ago
gradlew	using Gradle as build tool	4 months ago
gradlew.bat	using Gradle as build tool	4 months ago
settings.gradle	preparing release 2.7.0	2 months ago

full conversion CityGML <-> CityJSON

Compression factor == ~6X

file	CityGML size (original)	CityGML size (w/o spaces)	textures?	CityJSON	CityJSON compressed	compression factor
CityGML demo "GeoRes"	4.3MB	4.1MB	yes	582KB	524KB	8.0
CityGML v2 demo "Railway"	45MB	34MB	yes	4.5MB	4.3MB	8.1
Den Haag "tile 01"	23MB	18MB	no, material	3.1MB	2.9MB	6.2
Montréal VM05	56MB	42MB	yes	5.7MB	5.4MB	7.8
New York LoD2 (DA13)	590MB	574MB	no	110MB	105MB	5.5
Rotterdam Delfshaven	16MB	15MB	yes	2.8MB	2.6MB	5.4
Vienna	37MB	36MB	no	5.6MB	5.3MB	6.8

One example: Zürich LoD2 buildings



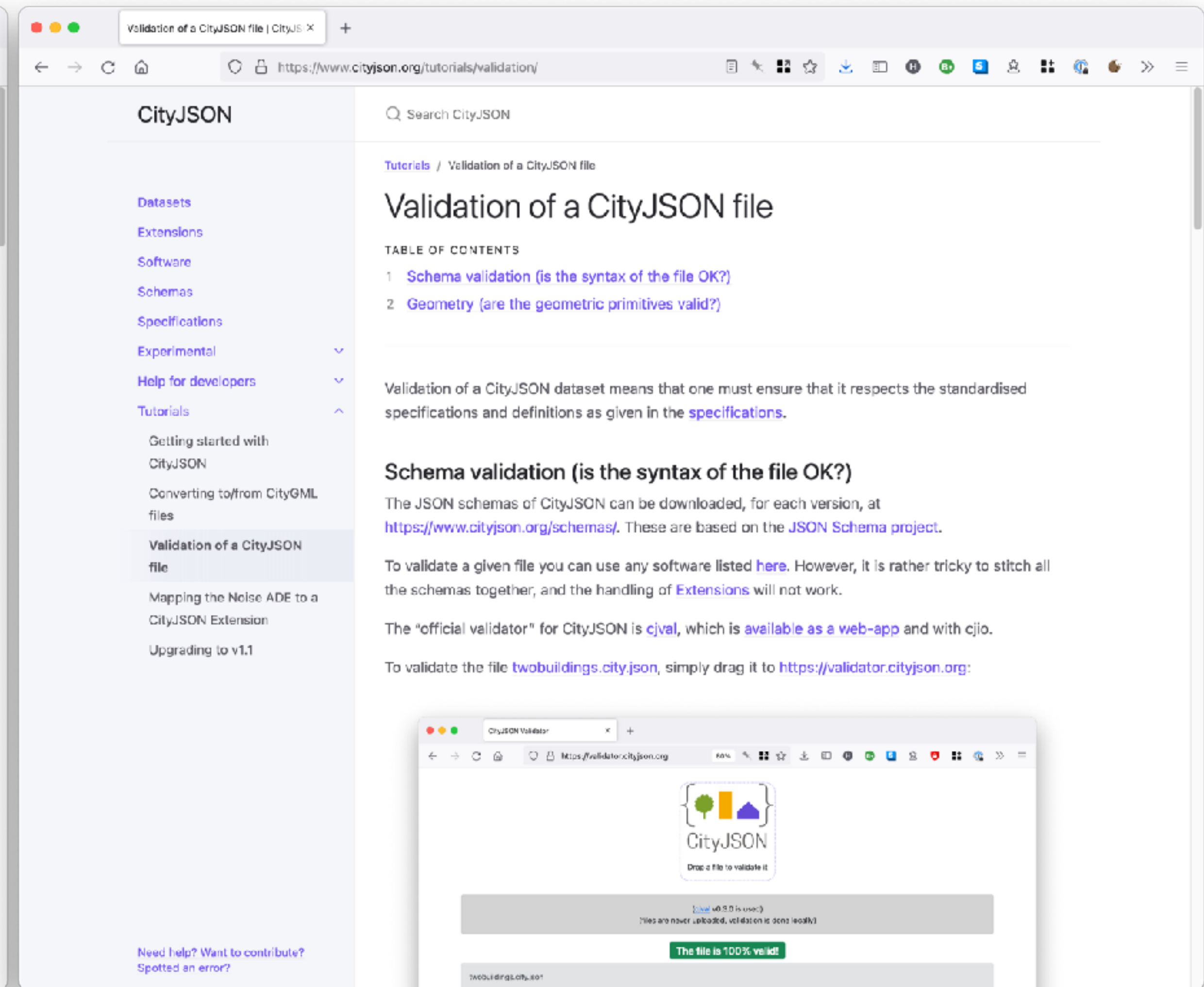
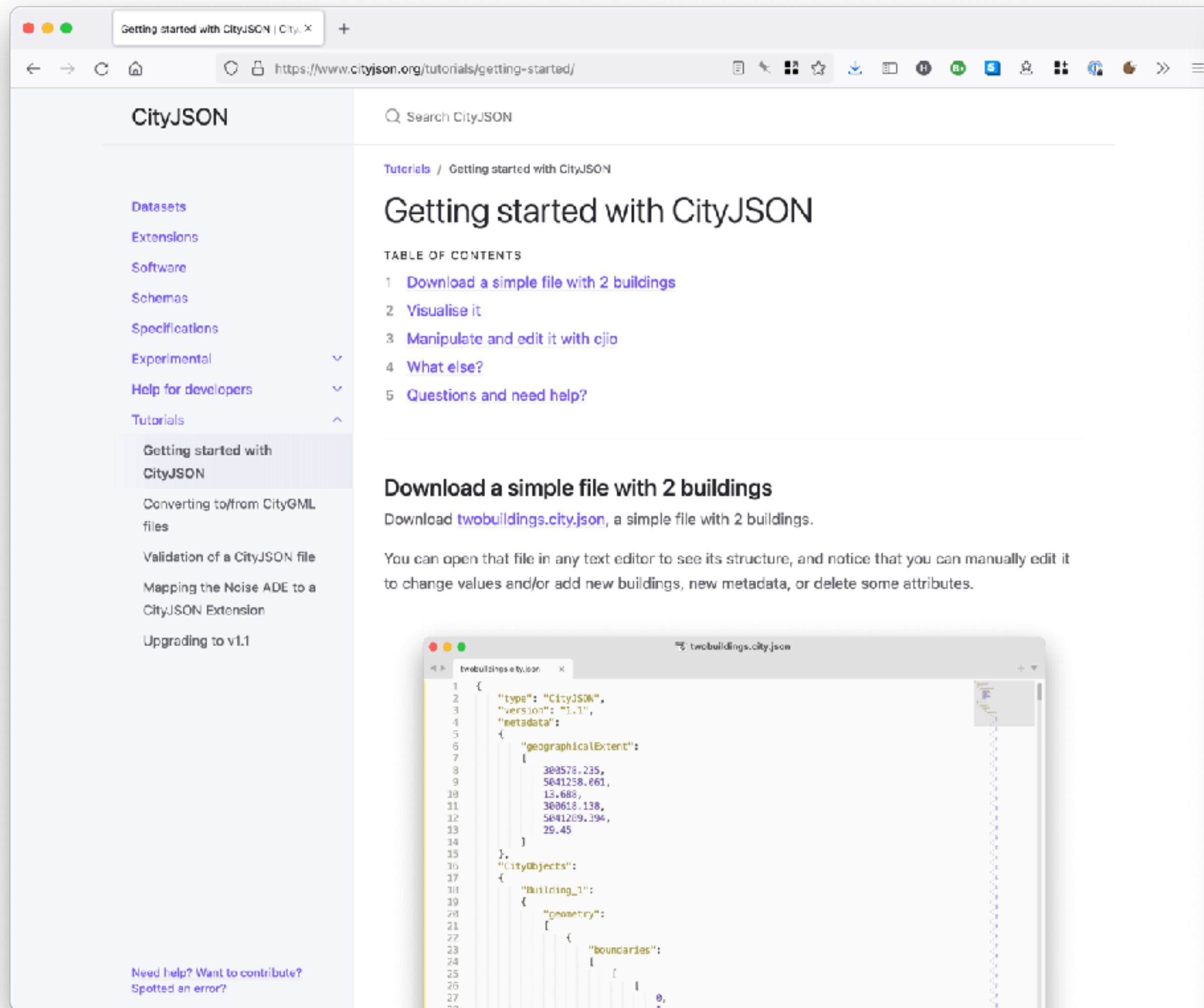
CityGML = 3.0GB

(but 1GB of spaces/CRs/tabs!)

CityJSON = 292MB

Compression == 7.1X

Getting started?



Filip Biljecki lecture about SVI

Thursday 11 June @ 13:45—14:45

Room F