CityJSON

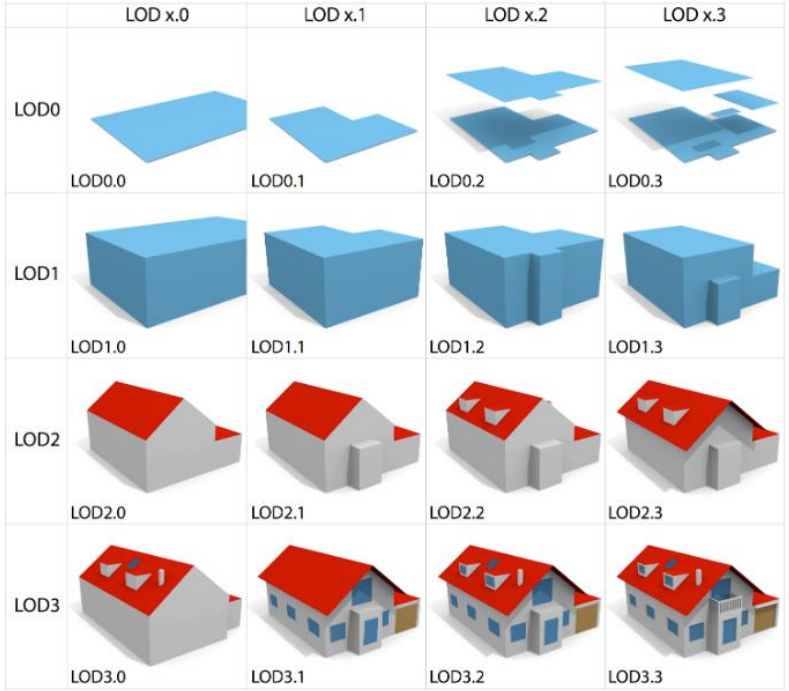
iiixr 김미송

CityJSON Overview

- CityJSON is a JSON-based encoding for storing 3D city models

- CityJSON defines way to describe most of the common 3D features and objects found in cities (such as buildings, roads, rivers, bridges, vegetation and city furniture)

- Also defines different standard levels of details(LODs) for the 3D object



What can be stored in CityJSON?

CityJSON mainly describes the geometry, attributes, and semantics of different kinds of 3D city objects

Types of objects stored in CityJSON

- **Appearance**: textures and materials for other types
- **Bridge**: bridge-related structures, possibly split into parts
- **Building**: the exterior and possibly the interior of buildings with individual surfaces that represent doors, windows, etc.
- **CityFurniture**: benches, traffic lights, signs, etc.
- **CityObjectGroup**: groups of objects of other types
- **Generics**: other types that are not explicitly covered
- **LandUse**: areas that reflect different land uses, such as urban, agricultural, etc.
- **Relief**: the shape of the terrain
- **Transportation**: roads, railways and squares
- **Tunnel**: tunnels, possibly split into parts
- **Vegetation**: areas with vegetation or individual trees
- **WaterBody**: lakes, rivers, canals, etc.

Coordinates of the vertices

- One vertex must be an array with exactly 3 integers, representing the (x,y,z) location of the vertex before it is transformed to its real-world coordinates.

```
"vertices": [
  [102, 103, 1],
  [11, 910, 43],
  [25, 744, 22],
  ...
  [23, 88, 5],
  [8523, 487, 22]
]
```

Arrays to represent boundaries

MultiPoint : Has an array with the indices of vertices

```
{
  "type": "MultiPoint",
  "lod": "1",
  "boundaries": [2, 44, 0, 7]
}
```

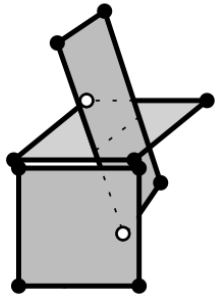
MultiLineString : Has an array of arrays, each containing the indices of a LineString

```
{
  "type": "MultiLineString",
  "lod": "1",
  "boundaries": [
    [2, 3, 5], [77, 55, 212]
  ]
}
```

Arrays to represent boundaries

MultiSurface : Has an array containing surfaces, each surface is modelled by an array of array.

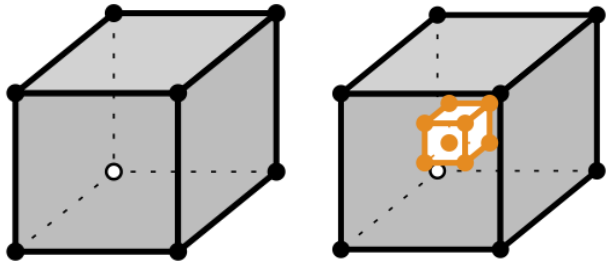
```
{
  "type": "MultiSurface",
  "lod": "2",
  "boundaries": [
    [[0, 3, 2, 1]], [[4, 5, 6, 7]], [[0, 1, 5, 4]]
  ]
}
```



MultiSurface

Solid : Has an array of shells, the first array being the exterior shell of the solid, and the others the interior shells. Each shell has an array of surfaces.

```
{
  "type": "Solid",
  "lod": "2",
  "boundaries": [
    //-- exterior shell
    [ [[0, 3, 2, 1, 22]], [[4, 5, 6, 7]], [[0, 1, 5, 4]], [[1, 2, 6, 5]] ],
    //-- interior shell
    [ [[240, 243, 124]], [[244, 246, 724]], [[34, 414, 45]], [[111, 246, 5]] ]
  ]
}
```



Shell Solid

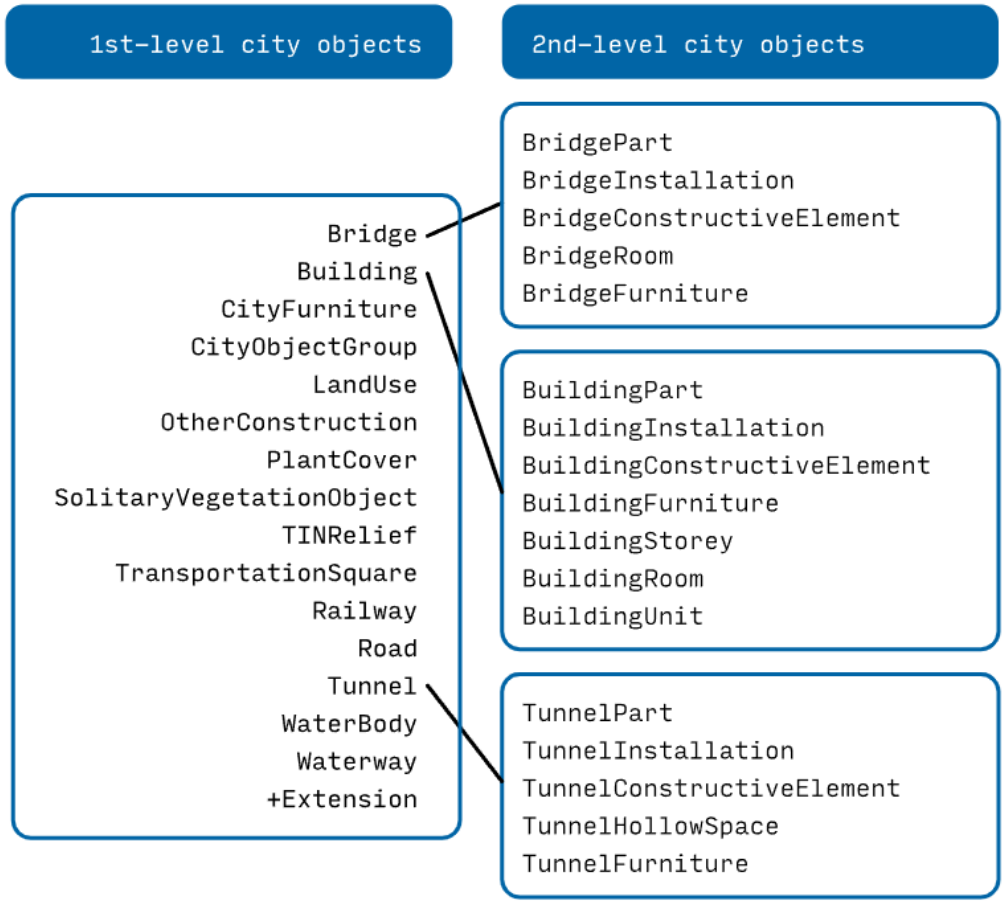
Semantics of geometric primitives

Specify surface properties

```
{
  "type": "MultiSurface",
  "lod": "2",
  "boundaries": [
    [[0, 3, 2, 1]],
    [[4, 5, 6, 7]],
    [[0, 1, 5, 4]],
    [[0, 2, 3, 8]],
    [[10, 12, 23, 48]]
  ],
  "semantics": {
    "surfaces" : [
      {
        "type": "WallSurface",
        "slope": 33.4,
        "children": [2]
      },
      {
        "type": "RoofSurface",
        "slope": 66.6
      },
      {
        "type": "+PatioDoor",
        "parent": 0,
        "colour": "blue"
      }
    ],
    "values": [0, 0, null, 1, 2]   # indexs of array "surfaces"
  }
}
```

The different City Objects

1. 1st-level : City Objects that can “exist by themselves”.
2. 2st-level : City Objects that need to have a “parents” to exist.



The different City Objects

Example 1 : Building

```
"CityObjects": {
  "id-1": {
    "type": "Building",
    "geographicalExtent": [ 84710.1, 446846.0, -5.3, 84757.1, 446944.0, 40.9 ],
    "attributes": {
      "measuredHeight": 22.3,
      "roofType": "gable",
      "owner": "Elvis Presley"
    },
    "children": ["id-2"],
    "geometry": [{...}]
  },
  "id-2": {
    "type": "BuildingPart",
    "parents": ["id-1"],
    "children": ["id-3"],
    ...
  },
  "id-3": {
    "type": "BuildingInstallation",
    "parents": ["id-2"],
    ...
  },
  "id-4": {
    "type": "LandUse",
    ...
  }
}
```

The different City Objects

Example 2 : Bridge

```
"CityObjects": {
  "LondonTower": {
    "type": "Bridge",
    "address": [
      {
        "City": "London",
        "Country": "UK"
      }
    ],
    "children": ["Bext1", "Bext2", "Inst-2017-11-14"],
    "geometry": [{
      "type": "MultiSurface",
      "lod": "2",
      "boundaries": [
        [[0, 3, 2, 1]],
        [[4, 5, 6, 7]],
        [[0, 1, 5, 4]],
        [[1, 2, 6, 5]],
        [[2, 3, 7, 6]],
        [[3, 0, 4, 7]]
      ]
    }]
  }
}
```

The different City Objects

Example 3 : WaterBody

```
"mygreatlake": {
  "type": "WaterBody",
  "attributes": {
    "usage": "leisure",
  },
  "geometry": [{
    "type": "Solid",
    "lod": "2",
    "boundaries": [
      [ [[0, 3, 2, 1]], [[4, 5, 6, 7]], [[0, 1, 5, 4]] ]
    ]
  }]
}
```

Transform Object

To reduce the size of a JSON object (and thus the size of files) and to ensure that only a fixed number of digits is stored for the coordinates of the geometries, the coordinates of the **vertices of the geometries are represented integer scales**.

A CityJSON object must therefore have one member “transform”, whose values are 2 JSON objects (“scale” and “translate”), both arrays with 3 values.

```
"transform": {
    "scale": [0.001, 0.001, 0.001],
    "translate": [442464.879, 5482614.692, 310.19]
}
```

Get real position of a given vertex v

v[0] = (vi[0] * ["transform"]["scale"][0]) + ["transform"]["translate"][0]
v[1] = (vi[1] * ["transform"]["scale"][1]) + ["transform"]["translate"][1]
v[2] = (vi[2] * ["transform"]["scale"][2]) + ["transform"]["translate"][2]

Reference System (CRS)

The coordinate reference system (CRS) may be given as a URL, formatted this way according to the OGC.
http://www.opengis.net/def/crs/{autority}/{version}/{code}

where {authority} designates the authority responsible for the definition of this CRS (usually "EPSG" or "OGC"), and where {version} designates the specific version of the CRS("0" is used if there is no version).

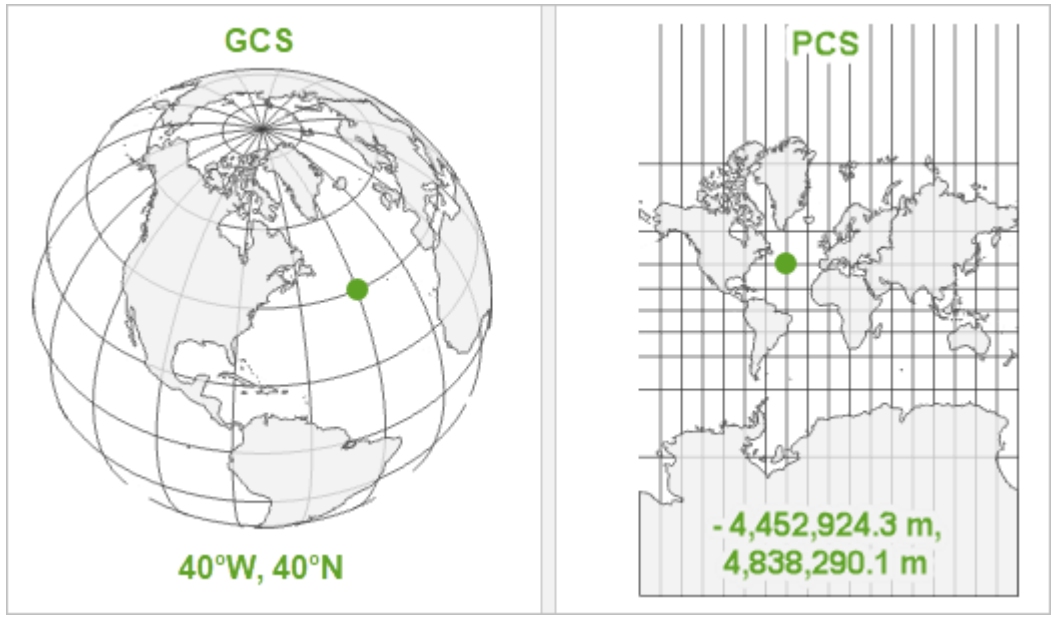
```
"metadata": {
  "referenceSystem": "https://www.opengis.net/def/crs/EPSG/0/7415"
}
```

* Coordinate Reference System(CRS)

CRS can be divided into **projected coordinate reference system** and **geographic coordinate reference systems**.

Geographic Coordinate Systems
: GCS use degrees of latitude and longitude and sometimes also a height value.

Projected Coordinate Reference Systems
: Projected CRS represents locations on the Earth using cartesian coordinates on a planar surface created by a particular map projection.



GeographicalExtent (bbox)

While this can be extracted from the dataset itself, it is often useful to store it.
It may be stored as an array with 6 values: [minx, miny, minz, maxx, maxy, maxz].

```
"metadata": {
  "geographicalExtent": [ 84710.1, 446846.0, -5.3, 84757.1, 446944.0, 40.9 ]
}
```

Example

```
data : {
    "type": "CityJSON",
    "version": "1.1",
    "metadata": {
        "referenceSystem": "https://www.opengis.net/def/crs/EPSG/0/7415"
    },
    "transform": {...},
    "CityObjects": {
        "id-1": {
            "type": "Building",
            "attributes": {
                "measureHeight": 22.3,
                "roofType": "gable",
                "owner": "Elvis Presley"
            },
            "geometry": [
                {
                    "type": "MultiSurface",
                    "lod": "2.2",
                    "boundaries": [
                        [[0, 3, 2, 1]], [[4, 5, 6, 7]], [[0, 1, 5, 4]]
                    ]
                }
            ]
        }
    },
    "vertices": [
        [23.1, 2321.2, 11.0],
        [111.1, 321.1, 12.0],
    ],
    "appearance": {
        "materials": [],
        "textures":[],
        "vertices-texture":[]
    }
}
```

All geometries have the same CRS

All City Objects listed here, indexed by their ID
Each have geometries + attributes

Geometry is ID of the vertex, global list

Material + Texture possible