V3ct3D - Report

Structuring project

Hugo BALTZ Samuel BENKIMOUN Victor BRINON
Elsa DARROMAN Hanane DERBOUZ
Sibawaih ER-RAZKI Hind HAMYA
Julie MARCUZZI Romain MAZIERE Loïc MESSAL
Rudolf MILLET Camille PARISEL
Mamady SAMASSA Anas SLIM

10 december 2016

Table des matières

1 Glossary

- **BDTopo** 3D IGN vector database of infrastructures with a metric precision.
- **BDUni** Internal global IGN database which is the source of many IGN product.
- **Bounding box** Enclosing box also call hyperrectangle of an geometric object.
- **BVH : Bounding Volume Hierarchy** Tree structure on a set of geometric objects. All geometric objects are wrapped in bounding volumes that form the leaf nodes of the tree.
- **B3DM : Batched 3D Model** Offline batching of heterogeneous 3D models for efficient streaming to a web client for rendering and interaction.
- **Cesium** An open-source JavaScript library for world-class 3D globes and maps.
- **GeoJSON** Format for encoding a variety of geographic data structures.
- **Geoportail** French public web portal to search and visualize services for geographical or geo-localized data.
- **glTF GL Transmission Format** Specification for the efficient transmission and loading of 3D scenes and models by applications.
- **IGN : Institut national de l'information géographique et forestière** French public institute which ensures the production,the maintenance and the dissemination of the geographic information in France.
- **ITowns** IGN technology platform for visualizing and exploiting 3D geographic data across the web.
- **LOD : Level of details** In client side, it is the decreasing of the complexity of a 3D model representation as it moves away from the viewer or according to other metrics such as object importance, viewpoint-relative speed or position.
- **Oslandia** Company specializing in GIS Open Source solutions.
- **OSM : Open Street Map** Project which aims to constitute a free geographical database of the world using the GPS system and other free data.
- **PostGIS** Plugin of the DataBase Management System PostgreSQL which activates the manipulation of geographic and spatial data.
- **Vector tile** Lightweight data format for storing geospatial vector data, such as points, lines, and polygons.

— **WMTS : Web Map Tile Service** Standard flow for serving pre-rendered georeferenced tiles over the Internet.

2 Introduction

2D mapping is omnipresent in our lives. For several years, 3D has been developed in research laboratories and the technology is ready. New application have emerged, especially with augmented and virtual reality. Indeed, the 3D facilitates the immersion in the space and it is important for many fields : urbanism, transport ...

We wish to bring a new dimension to cartography and geographic information by using existing databases containing architectural objects such as geolocalized buildings.

We have available IGN data that lists information on buildings on a national scale.

Originally developed for research at the IGN, Itowns is a digital globe that allows to view 3D data. The project has now moved into the Open Source world. Oslandia, an open source GIS solution company, is contributing to this project. This tool remains in development and improvement. It is destined to become the 3D visualization tool of the GeoPortail. Therefore, this transition requires a reflection on the data to be used as well as a modeling of the processes to be used. The interest of the 3D component within the geoportal is to allow, on the one hand, a user to have access to three-dimensional information and, on the other hand, to exploit a phenomenal quantity of this type of data.

Our work was to carry out a state of the art on the environment of 3D vector tiles to develop a production chain, from the creation of the tiles, from the data of the IGN, to their availability, in order to develop new services.

3 Modelisation

3.1 Use case diagrams

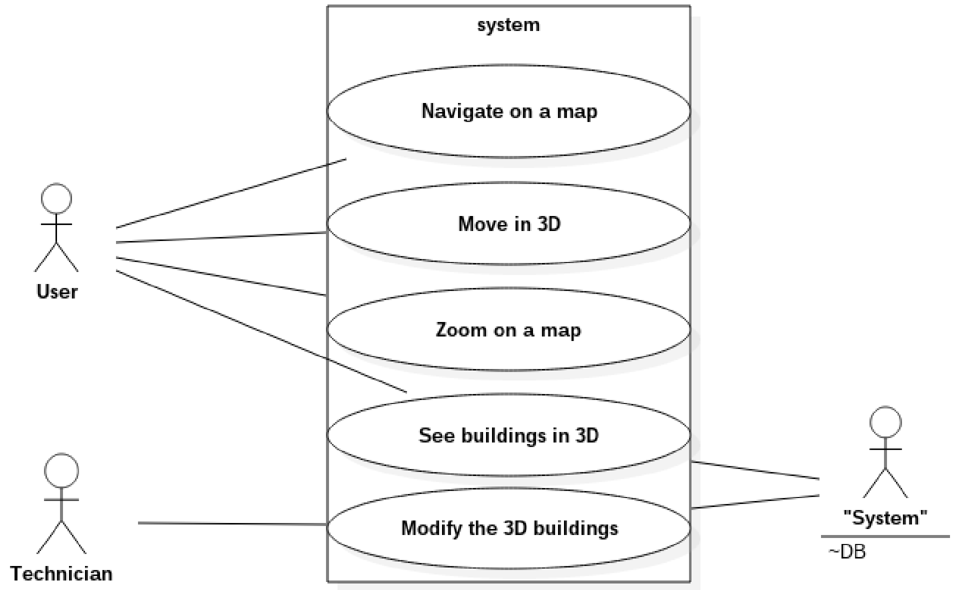
We identify two users for our application : * The general users * The technicians

Our application interacts with an external database that contains the 3DTiles

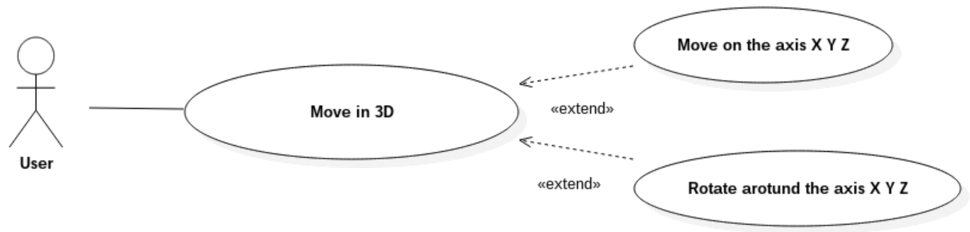
A user can navigate on a map, move in 3D, zoom on a map, see buildings in 3D.

A technician can modify the buildings.

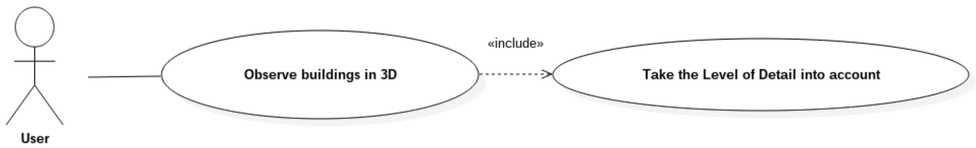
You can find this information in our use case diagram :



To move in 3D, the user can translate on the axis X,Y,Z and rotate around these axis :

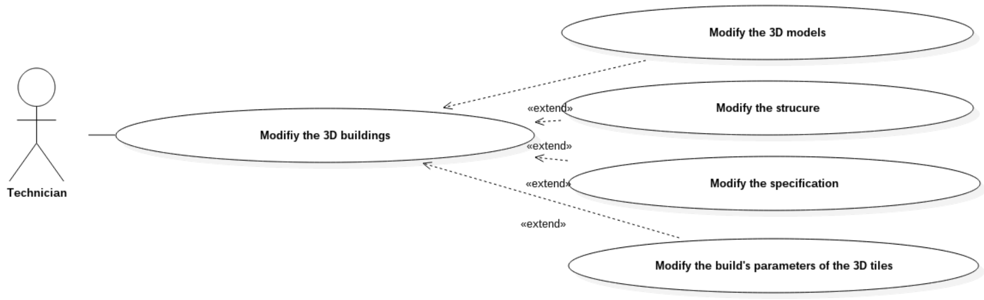


To see buildings in 3D, the system takes the Level of Details in account :



When the technician modifies a building, he can modify the 3D models , the structure, the specification of the building and the build's paramters of the

3D tiles :

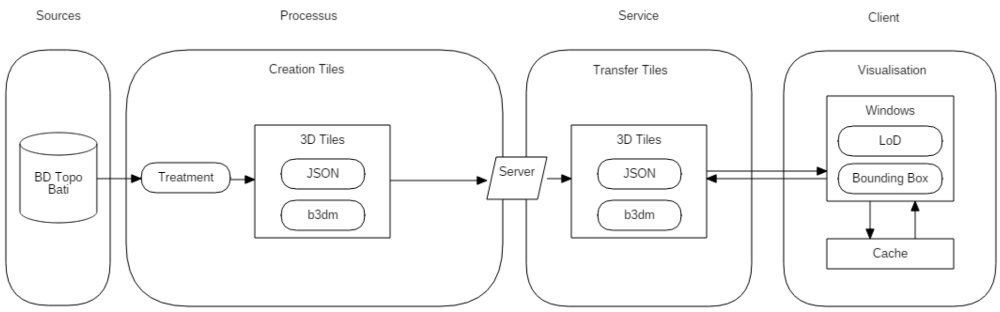


You can see our complete use case diagram in the annexes.

3.2 Production chain

3.2.1 Global vision

Following the various researches, we decide to set up a production chain describing the steps between the input data and the client part.



Our proposal of chain is to retrieve the 'Bati' data from the BD TOPO and collect all the information we need to create and complete our 3D Tiles in a BVH structure. They are composed of a JSON part and a B3DM part. They describ the properties of the objects and list technicals parameters of the visualisation. This 3D Tiles will be save in a server. All loaded information by a client will be stock in the cache part in the navigator, in case of reuse. The client part will make request, in order to obtain a visualisation of the data in the zone of interest, at the cache or at the server if the cache does not contain the information. This request takes in account the level of detail and the bounding box.

3.3 From BDTopo to a 3DTile server

In this part, we will focus on how to extract the relevant data from the building set in BDTopo and deal with missing information. The final goal is to fill all the required fields in a 3D Tile.

3.3.1 Description of BDTopo “Bati”

> **Source** : the BDTopo extract that we have used as a basis for our study is freely available at IGN, along with a 200pp document describing the source of data and the meaning of each data attributes.

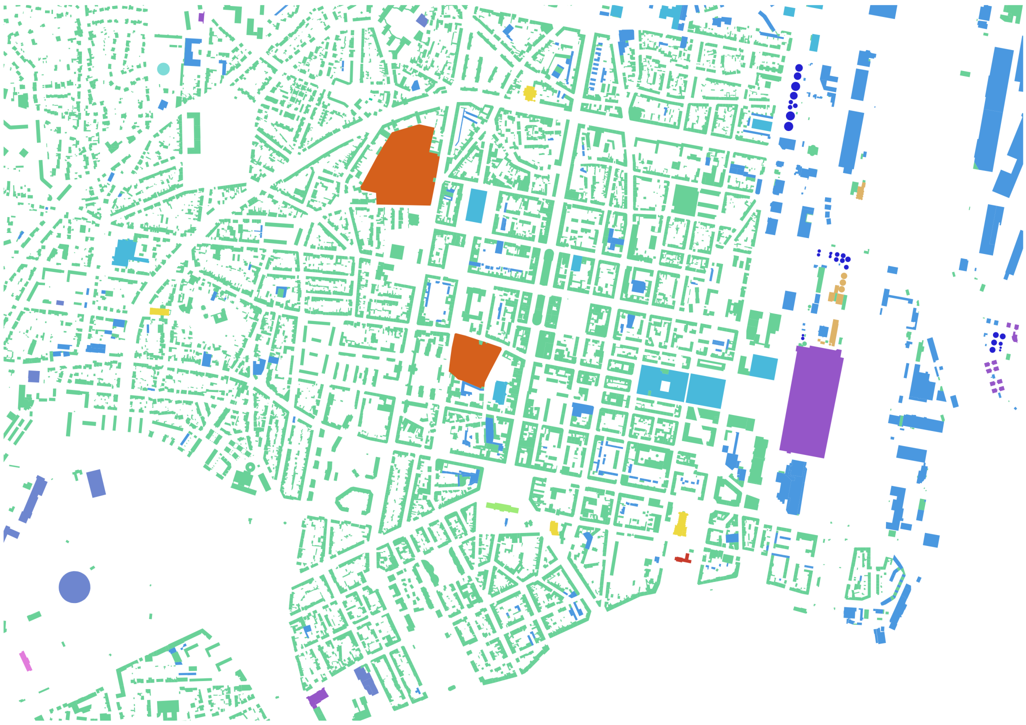
The **“E_BATI”** dataset contains 12 types of buildings (from “industrial” to “graveyard”). They all have the same set of attributes but, for simplicity we only focused on a few buildings. These are : **“BATI_INDUSTRIEL”**, **“BATI_REMARQUABLE”**, **“BATI_INDIFFERENCIE”**, **“CIMETIERE”**, **“LEGERE”**, **“RESERVOIR”**, **“TERRAIN_SPORT”**.

Definition of the attributes : - **ID** : unique id - **PREC_PLANI** and **PREC_ALTI** : respectfully for the precision in positioning and the precision in altitude. The latter depends on the data source (*cadastre* or other) - **ORIGIN_BAT** : the origin of the data (example : *cadastre*) - **HAUTEUR** : height of the building - **Z_MIN** and **Z_MAX** : minimum and maximum height at the gutter level (basically at the base of the roof). For “TERRAIN_SPORT”, these fields are replaced by **“Z_MOYEN”**.

> **Note** : The data imported from the *casdastre* has better 2D description and is more detailed than the other data (which have better 3D description and positioning) - *see p82 of BDTOPO_description manual version 2.2*

In addition to those fields, the BDTopo extract that we have provides a geometry : - **wkt_geom** : MultiPolygonZM data with (x,y) in the metric system (Lambert93), and z as the height of the building from sea-level. It provides a full description of the bounding upper surface of the object.

The illustration shows all the data available (from the subset of “E_BATI”)



3.3.2 What is a 3DTile ?

> **Note** : A thorough study of the strenght and current status of Cesium’s proposal regarding 3D Tile as a standard can be found in annex (see document : Summary of Cesium 3D tiles standard proposal). It also provides links to relevant Cesium webpages.

3DTile is an open source specification built on top of glTF (GL Transmission format). glTF is a very efficient way for transmitting and loading 3D content (as a binary stream). More information can be found in annex : Description of glTF.

3D Tile adds, among other things, spatial information and a hierarchy between objects.

In 3D Tiles, a **tileset** is a set of **tiles** organized in a spatial data structure, the **tree**. A tile references a **feature** or set of features, such as 3D models. The **metadata** for each tile - not the actual contents - are defined in JSON, as well as the tileset.

In summary, one tile description includes : - a tile.json with a bounding box definition, information necessary to handle a Hierarchical Level of Detail (HLOD). - a binary file with the description of the object and possibly some textures.

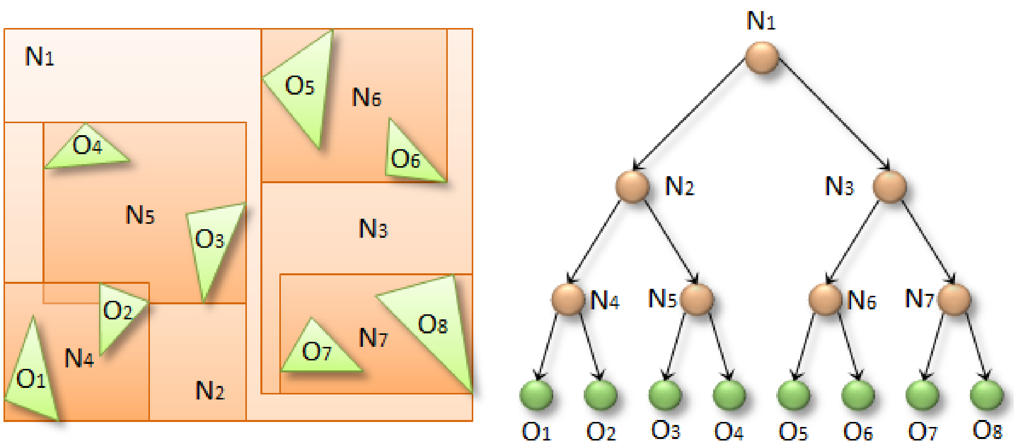
Among the possibilities, **Batched 3D Models** is the best way to describe a building. This format is described in annex : Description of B3DM.

3.3.3 Data hierarchization following the Bounding Volume Hierarchy method

3.3.3.1 Interest of the method

The main interest of using a Bounding Volume Hierarchy (BVH) in our workflow lays in the fastening of the geometric processing it allows. Indeed, it is essential that in an interactive web-mapping application, the graphic calculation times are reduced as much as possible, to make its using smoother. Mapping whole cities with their buildings, in 3D, implies complex et heavy data. This data shouldn't be displayed all at once for some obvious reasons of speed and memory performance.

Organizing data with a BVH tree, will rationalize the display, for each level of camera, by prioritizing the data to render. It also gives more cohesiveness to the information. For example, a small-scale point of view (with a weak zoom level) doesn't need to display small and minor geographic objects that would be barely visible and would overload the scene. Some objects also need to get their representation changed as the level of zoom changes too, to stay readable. If we zoom out, a building needs to get its shape generalized, meaning with less details and simplified shapes as the details wouldn't be visible anymore, and to lighten the data as more and more buildings are meant to be displayed as the view enlarges. Sometimes, some geographical objects could even need to be aggregated. The tree structure is, moreover, a far more efficient data organization. That makes each leaf of the tree (individual objects) or intermediate node (group of object) easily reachable via a database request. The complete dataset doesn't need to be read completely, as the right part of the tree can quickly be targeted.



3.3.3.2 Integration in the workflow

Our workflow follows the processes used by Oslandia, described in this article : http://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-2-W1/201/2016/isprs-annals-IV-2-W1-201-2016.pdf

When constructing the 3DTiles dataset, the tiles are linked with the hierarchy. On important thing to care about is the weight of each layer of hierarchy, that must be balanced in comparison with each other. The hierarchic level of each entity in the tree is determined according to its "weight". This weight, can recover multiple definitions depending on programmer choices, mostly in anticipation of who will visualize the data, and for what usage. By default, and without any further information of a possible professional usage, the biggest buildings are generally sent to the higher layers of the hierarchy. But the discretization could also be made according to certain building attributes for example.

In few words : a "weight formula" must have been decided before, to calculate the weight of each element, and assign him a level in the tree. Then, starting from our 3D Tiles database, the idea is to capture the bounding box enclosing the very first level of spatial elements (the most important ones), that will form the root of the tree. After, the tile has to be divided in 4 parts, that will generate the 4 first nodes. In each one of this nodes are assigned the bounding boxes of the entities belonging to the following rank to display. Finally, the parent bounding box (of the the superior level) is extended, so that it can contain the whole of the child bounding boxes. And the display hierarchy is made that way, once the parent bounding box is penetrated, the child bounding boxes start to be displayed. And so on, recursively.

The output must be a 3DTiles database, organized following this structure, ready to be hosted on a server.

3.3.4 Preliminary steps on BDTOPO data

In this part, we describe the necessary steps to build a 3D Tile object. First, some preliminary work has to be done to present the data in a usable format. The goal is to be able to automate as mush as possible the process.

3.3.4.1 Input Data : BDTopo

How to actually create a BDTopo building into a 3D Tile building BDTopo is a Shapefiles group with roads, energy network, hydrography, constructions, vegetation, etc...

A PostGIS database is created with all the building shapes into a unique table. Then, we can make build a set of SQL request to transform the IGN data into almost ready to use 3DTile data. For instance, a bounding box enclosing the object, coordinates in degrees, etc.

The BDTopo Bati entities geometry type is **MultiPolygonZM** : a 4D geometry, which is a 3D object (x,y,z) and m (for measurement) as a 4th dimension, which can be time or speed.

We simplified the geometry because the 4th dimension have no use for 3D Tiles, and transformed it into a simpler type : **PolygonZ**.

3.3.4.2 Importation into a PostGIS DB

The best way to import the shapefiles into the postgresql database is "shp2psql", which is in the *postgis* package.

```
shp2pgsql -S -s {SRID} -W "{encoding}" -a file.shp schema.table |
psql -d data_base -h host -U user
```

The created table for *bati* is :

```
CREATE TABLE topo_bati
(
  gid serial NOT NULL,
  id character varying(24),
  geom geometry(PolygonZ,2154),
```

```
  prec_plani double precision,
  prec_alti double precision,
  origin_bat character varying(8),
  nature character varying(255) DEFAULT NULL,
  hauteur smallint,
  z_min double precision,
  z_max double precision,
  CONSTRAINT topo_bati_pkey PRIMARY KEY (gid)
)
```

3.3.5 From a transformed BDTOPO to a set of 3DTiles

A 3DTile has one description file (in **json** format) and an object file (as a **B3DM** binary file). The *json* file tells the viewer where to display an object (geolocation), and when to display it (linked to a distance and a point of view). In addition, a **tileset.json** provides an overall description of all the 3D Tiles.

The transformed **BDTOPO** is ready to produce the following informations :

- the bounding box of each entity,
- the entity geometry,
- the entity placement with its bounding box.

The bounding box An object's bounding box is built with *ST_Envelope(geom)*. It must be in degrees system coordinates, so we have to transform the geometry into the target SRID.

The simplest is to build one bounding box per object as shown in the following illustration.

The entity We need the geometry in metric system, that is simple with data expressed in *Lambert 93*. A transformation matrix will provide the relative positioning of the geometry inside this bounding box.

The result format This illustration shows the available data (from the subset of “E_BATI”)

gid	geom	wkt_geom	hauteur	nature	first_point_x	first_point_y
	Object's geometry	*Object's wkt geometry*	*Object's height*	*nature*	*Object's x first point*	*Object's y first point*
168631	01030000206A...E5941	SRID=2154; POLYGON((305384.7 6699214,305384.1 6699210.9,305381.3 6699210.9,305380.6 6699215,305384.7 6699214))	6		305384.7	6699214

	bb_wkt_geom	bb_x_min	bb_y_min	delta_x	delta_y
	Bounding box's wkt geometry	*Bounding box x_min*	*Bounding box y min*		
	SRID=2154; POLYGON((305380.6 6699210.9,305380.6 6699215,305384.7 6699215,305384.7 6699210.9,305380.6 6699210.9))	305380.6	6699210.9	4.1	3.1

3.3.5.1 Tile metadata (json)

> The **boundingVolume.region** property is an array of six numbers that define the bounding geographic region with the order `[west, south, east, north, minimum height, maximum height]`. Longitudes and latitudes are in radians, and heights are in meters above (or below) the WGS84 ellipsoid. Besides region, other bounding volumes, such as box and sphere, may be used.

```
"boundingVolume": {
    "region": [
      -1.2419052957251926,
      0.7395016240301894,
      -1.2415404171917719,
      0.7396563300150859,
      0,
      20.4
    ]
  }
```

The rectangular bounding box (x,y coordinates) generated in the BDTOPO can be exported along with the **HAUTEUR** field which directly translate into `maximum height`.

In addition, the **boundingVolume** should be bigger than the real size of the building. A constant can be automatically applied.

```
"geometricError": 43.88464075650763,
```

> The **geometricError property** is a nonnegative number that defines the error, in meters, introduced if this tile is rendered and its children are not. At runtime, the geometric error is used to compute Screen-Space Error (SSE), i.e., the error measured in pixels. The SSE determines Hierarchical Level of Detail (HLOD) refinement, i.e., if a tile is sufficiently detailed for the current view or if its children should be considered.

In the BDTOPO extract, there is nothing regarding groups of buildings. For instance, in the case of a school with several buildings, we could build a 3D Tile with a single global volume and *children* tiles (see below) with a more precise volume for each building. The *mother* tile would get a **geometricError** referring to the approximation of the shape.

In our proposition, this value is not used.

```
"refine" : "add",
```

> The refine property is a string that is either “replace” for replacement refinement or “add” for additive refinement. It is required for the root tile of a tileset ; it is optional for all other tiles. When refine is omitted, it is inherited from the parent tile.

In our case, we will always set this field to **“add”**.

```
"content": {
    "boundingVolume": {
      "region": [
            ...
      ]
    },
    "url": "2/0/0.b3dm"
  },
```

> The **content** property is an object that contains metadata about the tile's content and a link to the content. **content.url** is a string that points to the tile's contents with an absolute or relative url. In the example above, the url, 2/0/0.b3dm, has a TMS tiling scheme, {z}/{y}/{x}.extension, but this is not required; see the roadmap Q&A.

"url" will point to the a bd3m file. The naming scheme comes from the Bounding Volume Hierarchy (BVH) process.

> **content.boundingVolume** defines an optional bounding volume similar to the top-level boundingVolume property. But unlike the top-level boundingVolume property, content.boundingVolume is a tightly fit bounding volume enclosing just the tile's contents.

We won't use **content.boundingVolume** even if it is easy to build - as it will always be similar to the **boundingVolume**.

In **content**, we can add metadata like the origin of the building data or the type of building. The viewer can then interpret that information and use a range of color to display this piece of information.

3.3.5.2 The actual object (B3DM)

Each B3DM file has a very similar header (**version**, **magic number**...) in addition to the length of the encoded data describing the object, which is in **glTF** format. Basically, one has to use the geometry (which describe a flat polygon) and the height of the building to generate a set of vertex points and normal vectors. Since glTF is a well established format it should be relatively easy to adapt an existing library with the BDTOPO inputs.

3.3.5.3 Tileset.json

The top-level object is basically a "super tile" encompassing all the other tiles. It has four properties : asset, properties, geometricError, and root.

```
"asset" : {
  "version": "0.0",
  "tilesetVersion": "e575c6f1-a45b-420a-b172-6449fa6e0a59"
},
```

This is purely for versioning purpose. ***

```
  "properties": {
    "Height": {
      "minimum": 1,
      "maximum": 241.6
    }
  },
```

properties.height is build from a minimum height (which can't be null) and the height of the highest object.

```
  "geometricError": 494.50961650991815,
```

As with tile.json files, this field value will be arbitrary set.

```
  "root": {
    "boundingVolume": {
      "region": [
        ...
      ]
    },
    "geometricError": 268.37878244706053,
    "content": {
      "url": "0/0/0.b3dm",
      "boundingVolume": {
        "region": [
        ...
        ]
      }
    },
    "children": [..]
  }
}
```

This part is similar to the one in **tile.json**. The **boundingVolume.region** is big enough to encompass all the object in the database. The **url** points to the highest level (see BVH) while the **children** section points to the next level **tile.json** files.

3.3.6 Server

All the files (json and B3DM) can be hosted in a database server and the queries will use the embedded BVH structure.

3.4 Visualization

In this section, we will describe the process and different parameters used in the visualization of 3d vector tiles.

3.4.1 Itowns

Oslandia had announced the first release of iTowns, a new 3D geospatial data visualization web framework developed by the iTowns project, including people from French IGN, Oslandia and AtolCD .

iTowns is a web framework written in Javascript/WebGL for visualisation of 3D geographic data, allowing precise measurements in 3D. Its first purpose is the visualisation of street view images and terrestrial lidar point clouds, though it now supports much more data types.

3.4.1.1 Technical basis

— JavaScript
— WebGL
— THREE.JS
— Shaders

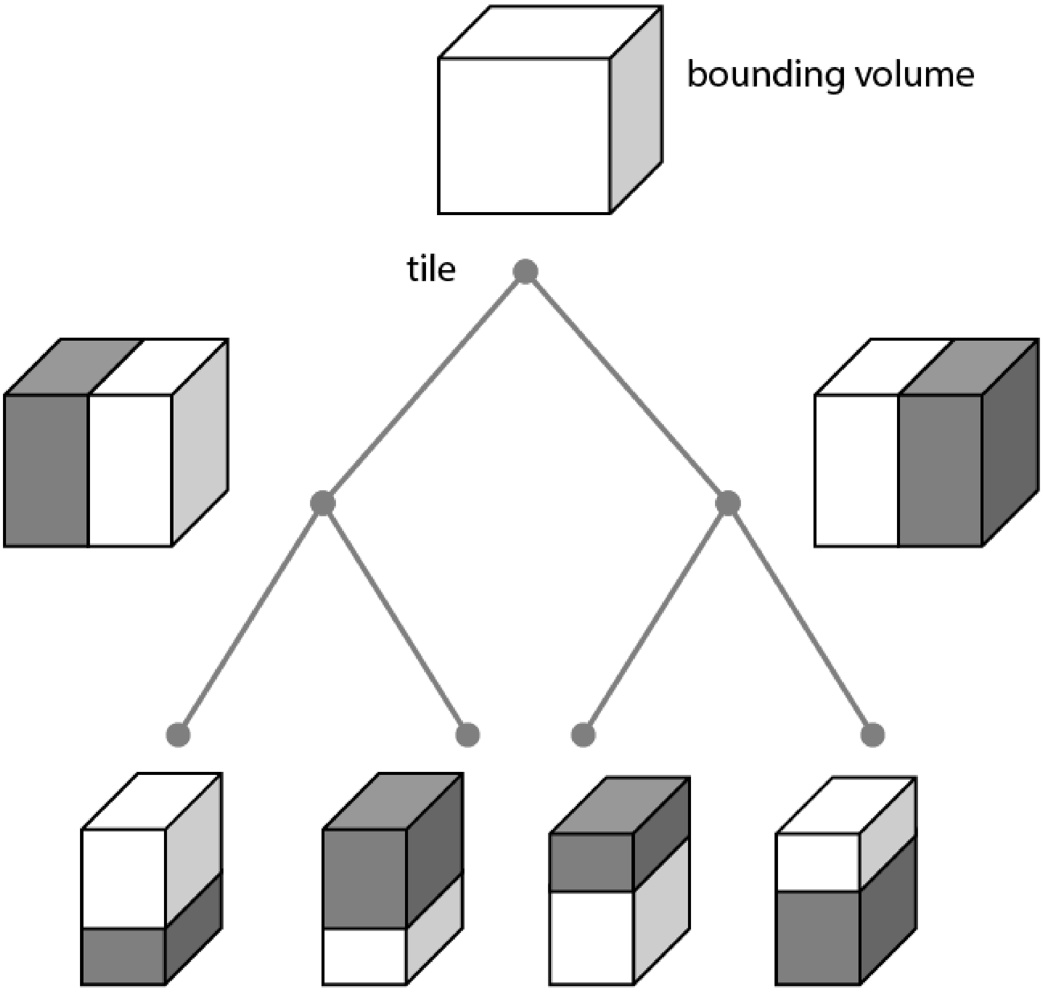
→ iTowns : client-side only

Oslandia is working on iTowns to support the GLTF format for 3d data display. Itowns uses a Javascript 3d library called THREE.JS which serves to load glTF format.

3.4.2 Tileset

Taken from : Tileset

A tileset is a set of tiles organized in a spatial data structure, the tree. Each tile has a bounding volume completely enclosing its contents. The tree has spatial coherence ; the content for child tiles are completely inside the parent's bounding volume. To allow flexibility, the tree can be any spatial data structure with spatial coherence, including k-d trees, quadtrees, octrees, and grids.



3.4.3 LOD

In client side, Level of detail involves decreasing the complexity of a 3D model representation as it moves away from the viewer or according to other metrics such as object importance, viewpoint-relative speed or position. Level of detail techniques increase the efficiency of rendering by decreasing the workload on graphics pipeline stages, usually vertex transformations.

If you want more inforamtion about the Level of Details see the annexe

3.4.4 Process

When the user launches iTowns, The view is initialized by a global tileset. The zoom level determines a bounding box of a tileset.

Each tileset is characterized by an identifier which is included in the request with the level of detail LOD.

The response to the request is a 3dtiles format which contains a gltf file that is required for display and can be loaded using the three.js library : GLTFLoader.js

The cache contains a copy of the original data when it is expensive to retrieve compared to cache access time. Once some 3d vector tiles are stored in the cache, the client can access them directly through the cache rather than retrieving them by requests, which reduces the access time.

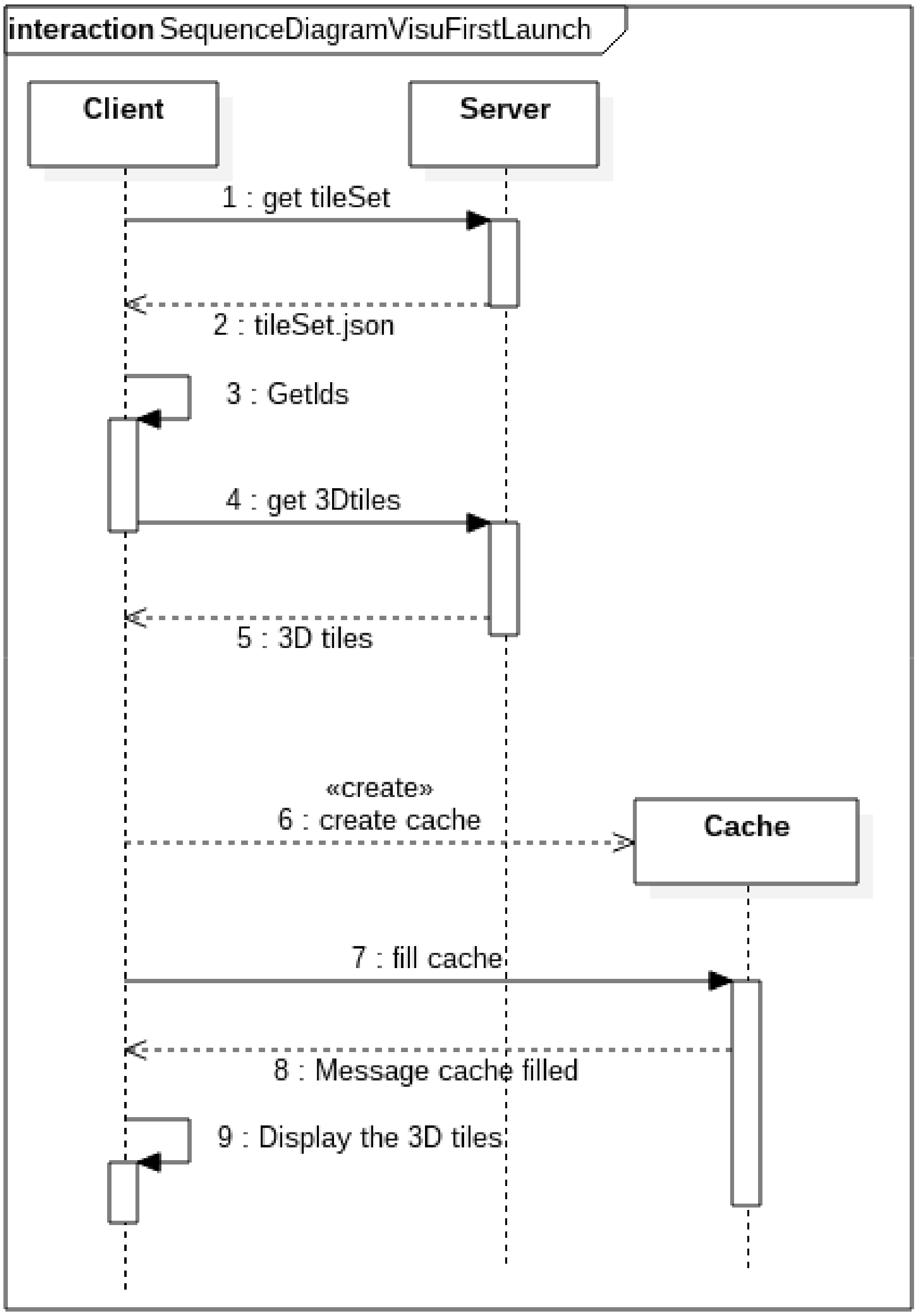
At this moment, the user can visualize the 3d tiles fluently with the desired level of detail due to cache system.

3.4.5 Sequence diagrams

Here, we will present how we imagine the opening of the data

3.4.5.1 On the first launch

This is the sequence diagram that presents how the application reacts on the first launch :



First the client sends a request to the server to get the tileSet.json that describes how the data are cut.

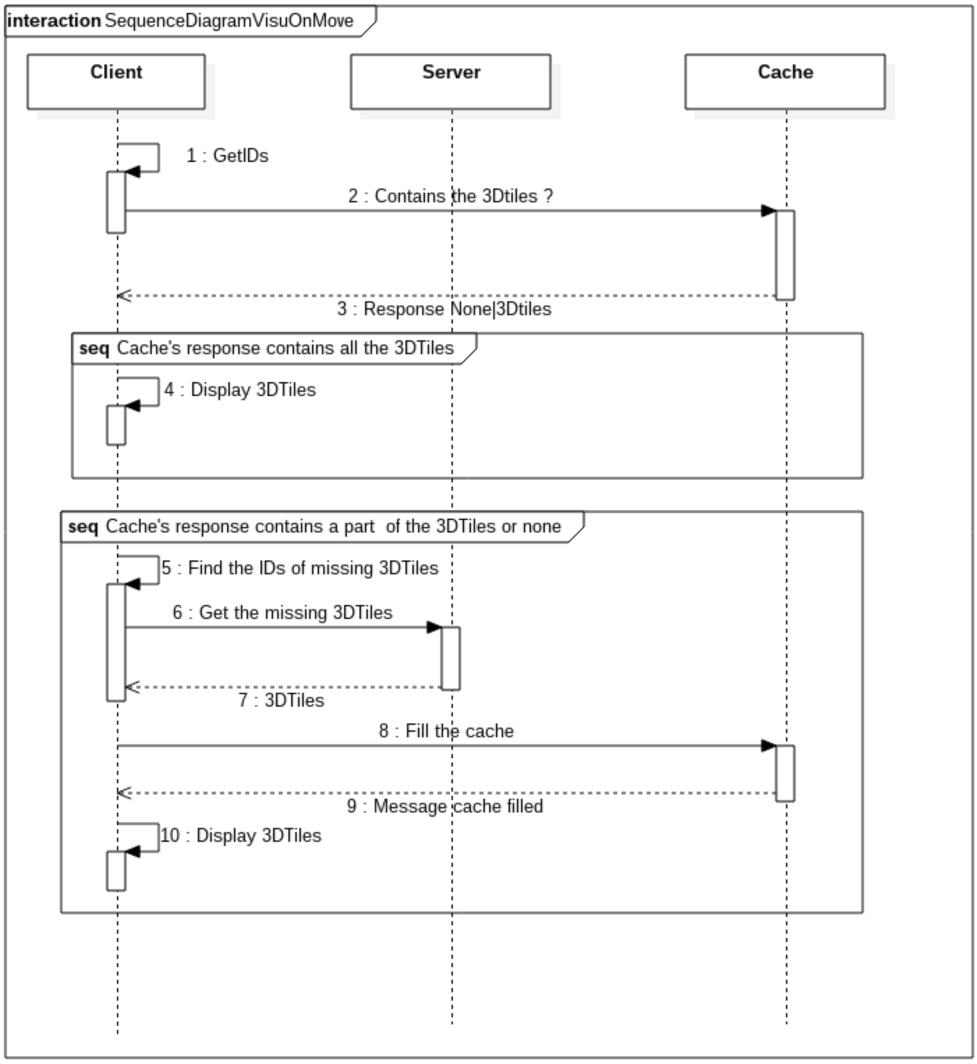
The using the bounding box of the screen, the client finds the IDs of the 3DTiles that he needs to ask to the server. When he recoveres the IDs, he sends a request to the server. The server's response is the 3DTiles that match the IDs.

Thererupon the client creates the cache and fills it with the 3DTiles received by the server.

Finally the client displays the 3DTiles on the screen.

3.4.5.2 On move

This is the sequence diagram that presents how the application reacts when the user moves on the map :



When the user moves on the map, the client finds the IDs of the 3DTiles that he needs to ask to the server. When he recoveres the IDs, he interogates the cache to know if it contains all, some or all the 3DTiles.

If the cache contains all the 3DTiles, the client displays these tiles.

Else, the client recovers the IDs of the missing 3Dtiles and sends a request to the server. The server's response is the 3DTiles that match the IDs. Thererupon the client fills the chache with the 3DTiles received by the server. Finally the client displays the 3DTiles on the screen.

4 Perspective : 4D Tile

In 3D, we master space with height, width and depth. But if we want to touch the 4D, we need to add a time component.We must pass into an imaginary referential of space-time. In physics, we must think about the evolution of matter and object as a function of time or duration. We do not have actually a efficient management of time on object.