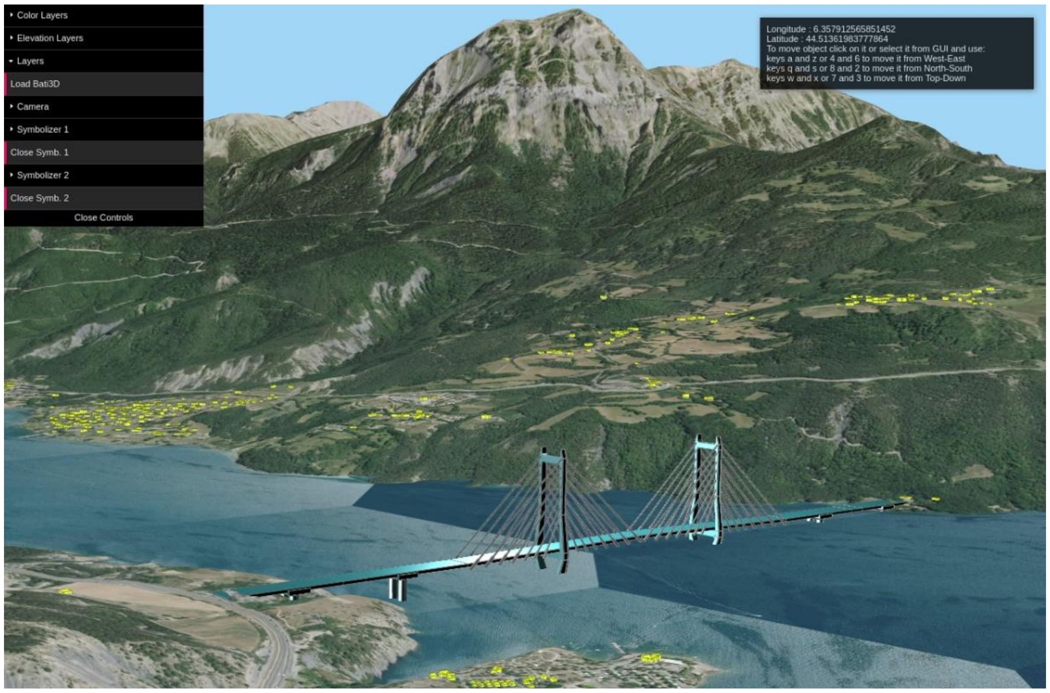
VISUALIZATION IN ITOWNS OF BUILDINGS ELEGANTLY STYLIZED



Longitude : 6.357912565851452
Latitude : 44.51361983777864
To move object click on it or select it from GUI and use:
keys a and z or 4 and 6 to move it from West-East
keys q and s or 8 and 2 to move it from North-South
keys w and x or 7 and 3 to move it from Top-Down

Authors

Houssem Adouni, El-Hadi Bouchaour, Arnaud Grégoire, Rose Mathelier, Laurie Nino, Adel Ouhabi, Ludivine Schlegel

Project backers

Mathieu Bredif, Sidonie Chistophe, Alexandre Devaux

ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES

INSTITUT NATIONAL DE L'INFORMATION GÉOGRAPHIQUE ET FORESTIÈRE

CONCEPTION OBJET ET GÉNÉRALISATION DE L'INFORMATION TOPOGRAPHIQUE

SUMMARY

INTRODUCTION

CONTEXT

The concertation in urban projects requires visual supports to ensure a correct communication between the different parts. These visual supports are often limited to the official documents, such as PLU, which are not very clear to the general public, and therefore unsuitable for an efficient communication strategy.

To create efficient and accessible data visualization tools is a recurrent problematic in geomatics, and particularly in cartography. It requires to consider technical factors – visualization factors, render techniques – but also human factors: how the human eye will perceive a visualization? How will it be interpreted? How could the data visualization be improved to make the human comprehension more efficient?

This problem is even wider when it comes to 3D data such as those manipulated in urban projects. Indeed, it implies to choose an adapted 3D data visualization platform, accessible for non-technical professional profiles, easy-to-use, efficient, and offering a wide range of possible stylization.

GOALS OF THE PROJECT

The VIBES (Visualization in iTowns of Buildings Elegantly Stylized) project consists in implementing geovisualization techniques to stylize buildings on the platform iTowns. This project aims to provide a visual support for city planning, among other purposes.

The user should be able to:

- Load one or several 3D files (of various formats).
- Transform its/their visual aspect(s) using a GUI.
- Save the current style in a JSON-like format (to be defined).
- Load an existing style to re-apply it, including predefined styles (transparent, typical, sketchy).

This project was proposed by Sidonie Christophe, from the COGIT laboratory (IGN), with Alexandre Devaux and Mathieu Bredif as technical support on iTowns.

PROJECT MANAGEMENT

This project will be carried out in March and April 2018 by a group of seven students in ENSG TSI, using SCRUM methodology. It will be divided into seven sprints, each one during a week.

TEAM

The seven members of the team are:

- Houssem Adouni
- El-Hadi Bouchaour
- Arnaud Grégoire
- Rose Mathelier *(Scrum Master)*
- Laurie Nino
- Adel Ouhabi
- Ludivine Schlegel

Given the number of people in the team, it is crucial to apply an efficient organization so everyone can be involved. To that end, we chose to work mostly in variable pairs, and to divide the tasks each week among these pairs (one person would be working alone since we are an uneven number).

BACKLOG

This is the planning we followed during the time of the project:

- **Sprint 1**: analysis, conception, first version of the tool, CI/CD.
- **Sprint 2**: architecture set-up, definition of the 3D style with basic parameters + texture on faces, saving and loading.
- **Sprint 3**: adaptation of the architecture to stylize several objects, first trials on shadow management and edge texturation, analyse of existing situation regarding BATI3D loading in iTowns, unit tests.
- **Sprint 4**: implementation of a BATI3D loader, edge stylization (dashed or continuous), basic geolocation and object movements, basic shadow management, layer management.
- **Sprint 5**: implementation of a BDTOPO loader (WFS), integration of the BATI3D loader in the project, edge textures (sketchy style), environment stylization (lights, shadows), possibility to click on objects.
- **Sprint 6**: correction of bugs, camera management, and report.
- **Sprint 7**: finalization, presentation.

A first provisional backlog was created at the beginning of the project. Each Friday afternoon, at the end of the sprint, we take some time for:

- a review of the work completed during the week.
- a backlog grooming to plan the next sprint and divide the tasks (usually one or two tasks per pair).
- updating the report.

MANAGEMENT TOOLS

- **GitHub** - for code hosting.
- **Travis** - for continuous integration.
- **Easybacklog** - for backlog and planning.
- **Trello** - for task assignment and file sharing.

ANALYSIS OF THE EXISTING SITUATION

THE ITOWNS ENVIRONMENT

The main challenge of this project is that it must be included into the architecture of the existing **ITOWNS**[1] project (version 2.3.0). Therefore, a necessary step is to get to know this architecture and to analyze it to know where our new functionalities could be located. (See **ANNEX 1**)

Two sorts of development can be carried out in iTowns:

- develop a new example, based on the existing classes of the core of iTowns.
- add new functionalities directly to the core.

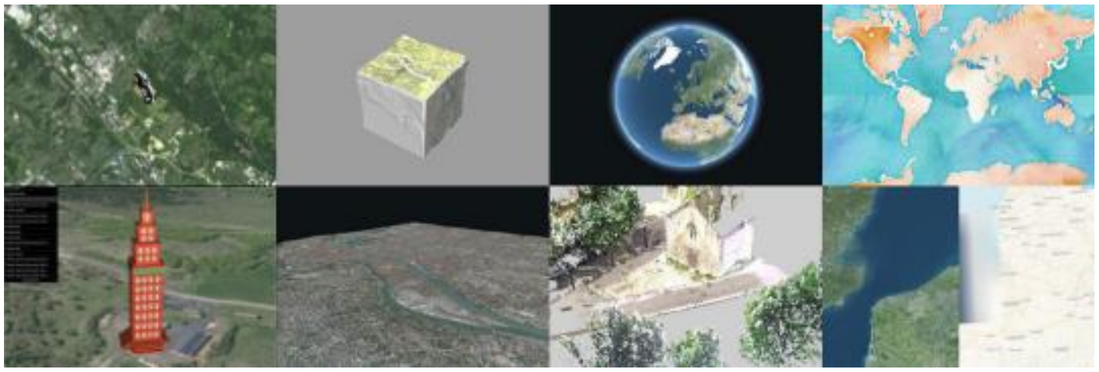


FIGURE 1: ITOWNS EXAMPLES

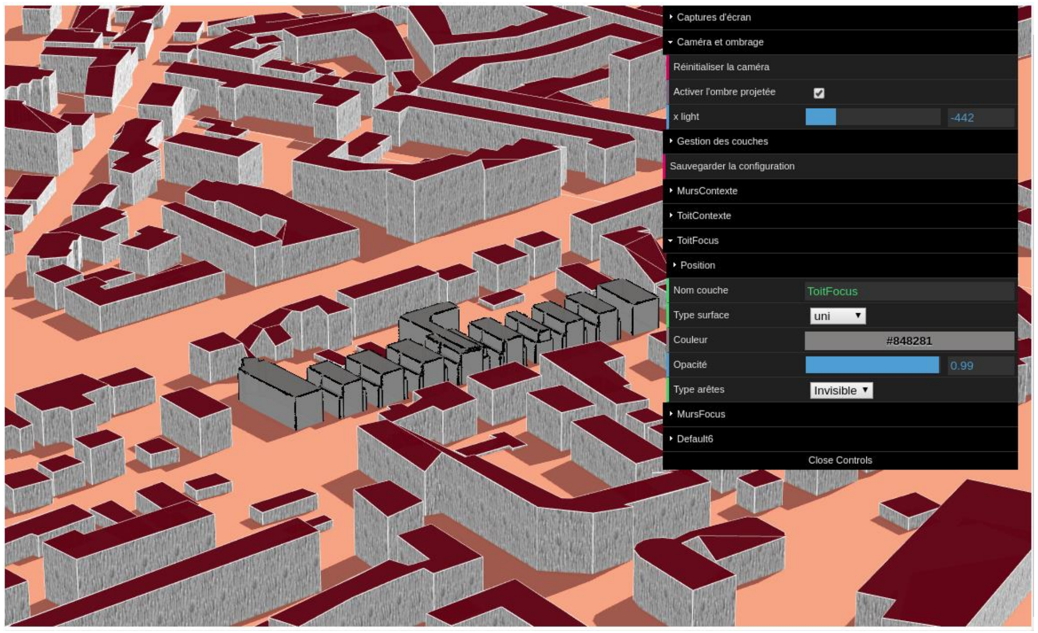
This choice depends on the purpose of the tool. Our stylization tool is intended to be applied in multiple examples, therefore the main functionalities should be integrated in the source of iTowns. This implies that they should be as generic as possible and respect the iTowns standards. An example will also be created, only to demonstrate how our tool should be used, but the goal is to make this example as simple as possible and to avoid including too much logic in it.

PLU++

The PLU++ project, developed in 2016 by Anouk Vinesse under the supervision of Sidonie Christophe and Mickaël Brasebin (COGIT), is a tool of 3D building stylization using Three.js, It will be used as proof of concept to help start our project. To that end, we analyzed the code from the latest version available on GitHub: **IGN/PLU2PLUS**[2]

[1] http://www.itowns-project.org/

[2] https://github.com/IGNF/PLU2PLUS



Activer l'ombre projetée ✓

x light -442

Nom couche ToitFocus

Type surface uni

Couleur #848281

Opacité 0.99

Type arêtes Invisible

FIGURE 2: VIEW OF PLU++

The goal of this analysis is to find out how the following things can be done:

- Loading an OBJ file.
- Applying a style to a mesh.
- Changing this style dynamically using a user interface (dat.GUI).

DESCRIPTION OF THE PROJECT

The project consists in a web page which display a 3D scene that contains a plane geometry with buildings on it, and a user menu made with dat.GUI to change the visual aspect of the buildings (color, opacity, style...) and do some other actions, such as re-center the camera, save the current style, etc.

There are two sorts of buildings:

- **Focus**: the main buildings the user wants to work on, loaded from an OBJ model.
- **Context**: the surrounding buildings, loaded from BDTOPO or BATI3D (also OBJ models).

For more details, see the **PRESENTATION OF PLU++**[3].

[3] http://ignf.github.io/PLU2PLUS/

STRUCTURE OF THE CODE

The main code is divided into 5 files, as follow:

- **rendu.html**: contains the main work.
- **fonctions_gui.js**: contains main functions for file loading, GUI initialization and update.
- **fonctions_load.js**: contains the utils functions to load 3D files.
- **fonctions_sliders.js**: contains the utils functions to handle the sliders.
- **fonctions.js**: contains other utils functions.

HOW DOES IT WORK?

1. The user chooses a JSON file as an input, to define the initial style.
2. The GUI is created with those initial parameters.
   (the style - discrete, typical, sketchy - determines which parameters of the GUI are visible).
3. Three.js materials are initialized with the initial parameters.
4. Event listeners are created on the GUI: each change will be displayed directly on the materials previously created (for the context) or on each vertex created (for the focus).
5. OBJ models are loaded with the current material.

POSSIBLE AMELIORATIONS

Although the PLU++ project successes in creating an easy-to-use interface to dynamically change the stylization of 3D objects through various parameters, its implementation has some limitations. Particularly, its structure does not clearly separate the 3D geometry and the stylization itself.

However, it provides a helpful set of functions that can be re-used in our project, particularly for edges extraction and texture application.

Therefore, the idea of our project is to re-make the concept of PLU++ inside the iTowns structure, but in a more generic way so it can be re-used and re-adapted more easily.

DEFINITION OF A STYLE

The definition of the term style depends on the field of research. In cartography, the term style is defined by an identification of visual characteristics on a map. The style is a way to show different types of render. The result can be shown in 3D thanks to the position of a virtual camera and light sources. A material is attributed to each object to determinate the appearance in the final render.

VISUAL VARIABLES ON CARTOGRAPHIC REPRESENTATION

In the book *Sémiologie Graphique* written by Jacques Bertin in 1967, the style is defined as a graphical expression of an information. The author describes, in his book, visual variables as the ways to manage the symbology of the 3D object on the scene. Bertin defined seven initial variables: **position**, **size**, **shape**,

orientation, **color hue**, **color value** and **texture**. These variables can help the viewer to have a simple idea of how the object is represented in a map. It is why they will be included in the project to manage the style of 3D object.

3D BUILDINGS STYLIZATION

In the iTowns project, the buildings style is described as a photo-realistic representation. The 3D object will look like as well as possible to the reality. The goal of the VIBES project is to modify the style of building and show them as an abstract representation. The abstract representation can be described with different 'generic' styles:

- **Discrete**: this style displays very fine or dashed lines and transparency.
- **Typical**: this style is similar to the hues of real buildings with characteristic elements like door, windows or chimney. It looks like photo-realistic style.
- **Sketchy**: this style represents buildings as if they were drawn with pastel and bright colors.

These 'generic' styles were described by Anouk Vinesse in the PLU++ project.

CONCEPTION AND DEVELOPMENT

STYLE FORMAT

In order to save and re-use the style of an object, we need to find a way to store this information, so it can be accessed easily. The obvious solution, in a JavaScript project, is JSON.

The next question is: what do we stylize? Does the stylization concern a single mesh, or a complex object with several styles at once? A concertation with the client led us to implement both alternatives: two stylization methods will be created.

Therefore, there will also be two types of style formats:

- Generic style, applicable to any mesh:

```
{
   "edges":{
      "opacity":1,
      "color":"#ed0606",
      "width":2.525429673798667,
      "style":"Dashed",
      "dashSize":0.2877972641178534,
      "gapSize":0.05
   },
   "faces":[
      {
         "opacity":1,
         "color":"#dadeda",
         "emissive":"#938383",
         "specular":"#111111",
         "shininess":67.34479130129779,
         "texture":"./textures/bricks.jpg",
         "textureRepeat":0.374991231146966
      }
   ]
}
```

- Style format for a complex object with several meshes, all defined by a name:

```json
{
  "edges": {
    "opacity": 1,
    "color": "#000000",
    "width": 0.5050859347597334,
    "style": "Continuous"
  },
  "faces": [
     {
      "name": "Tower1_Tower",
      "opacity": 1,
      "color": "#a06b6b",
      "emissive": "#614848",
      "specular": "#111111",
      "shininess": 31.55357142857143,
      "texture": "./textures/bricks.jpg",
      "textureRepeat": 0.1
    },
    {
      "name": "Tower1_window",
      "opacity": 1,
      "color": "#ffffff",
      "emissive": "#d4a9a9",
      "specular": "#f2dbdb",
      "shininess": 30,
      "texture": "./textures/glass.png",
      "textureRepeat": 1.5901250000000002
    },
    {
      "name": "Tower1_Tower_Fence_posts_Tube_1",
      "opacity": 1,
      "color": "#ffffff",
      "emissive": "#000000",
      "specular": "#111111",
      "shininess": 30,
      "texture": null,
      "textureRepeat": null
    }
  ]
}
```

3D STYLIZATION PROCESS

The 3D stylization will be done according to the following activity diagram:

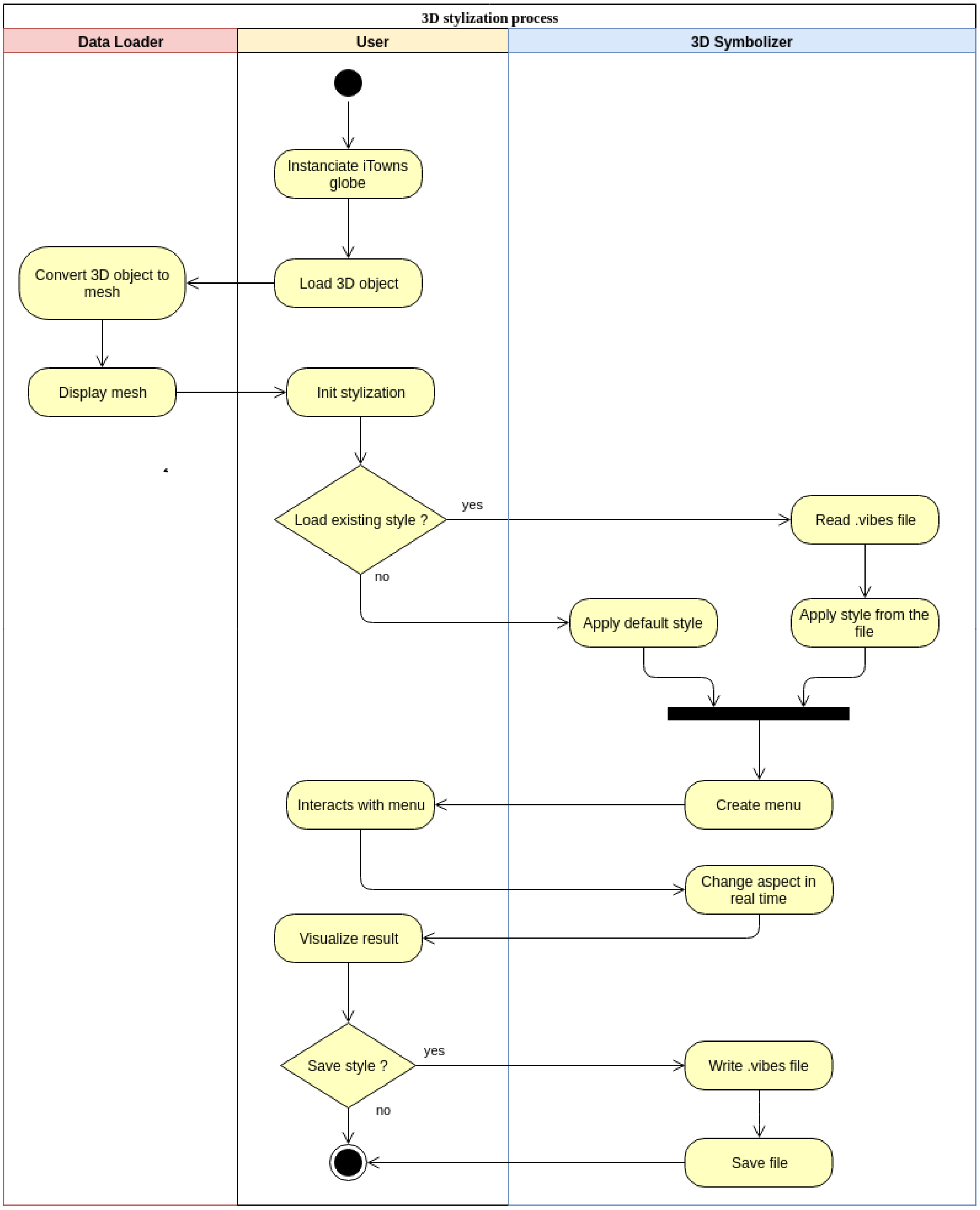


FIGURE 3: ACTIVITY DIAGRAM OF THE STYLIZATION PROCESS

ARCHITECTURE

GLOBAL ARCHITECTURE

The architecture of our project must be included in iTowns. The schema in **ANNEX 2** shows the different functionalities of iTowns, with the ones that interest us in red:

The goal is to make this tool as general as possible, which means it must not depend on just one example. On the contrary, it should be usable on any example containing a 3D object on an instance of the globe, as a full-fledged functionality of iTowns. Therefore, we created a new class Symbolizer, which manages the 3D render. We also extended the loading functionalities of iTowns in order to handle '.obj' files and other formats, using a new class called ModelLoader. These two classes are called by another class called LayerManager.

The final architecture of our project is the following:

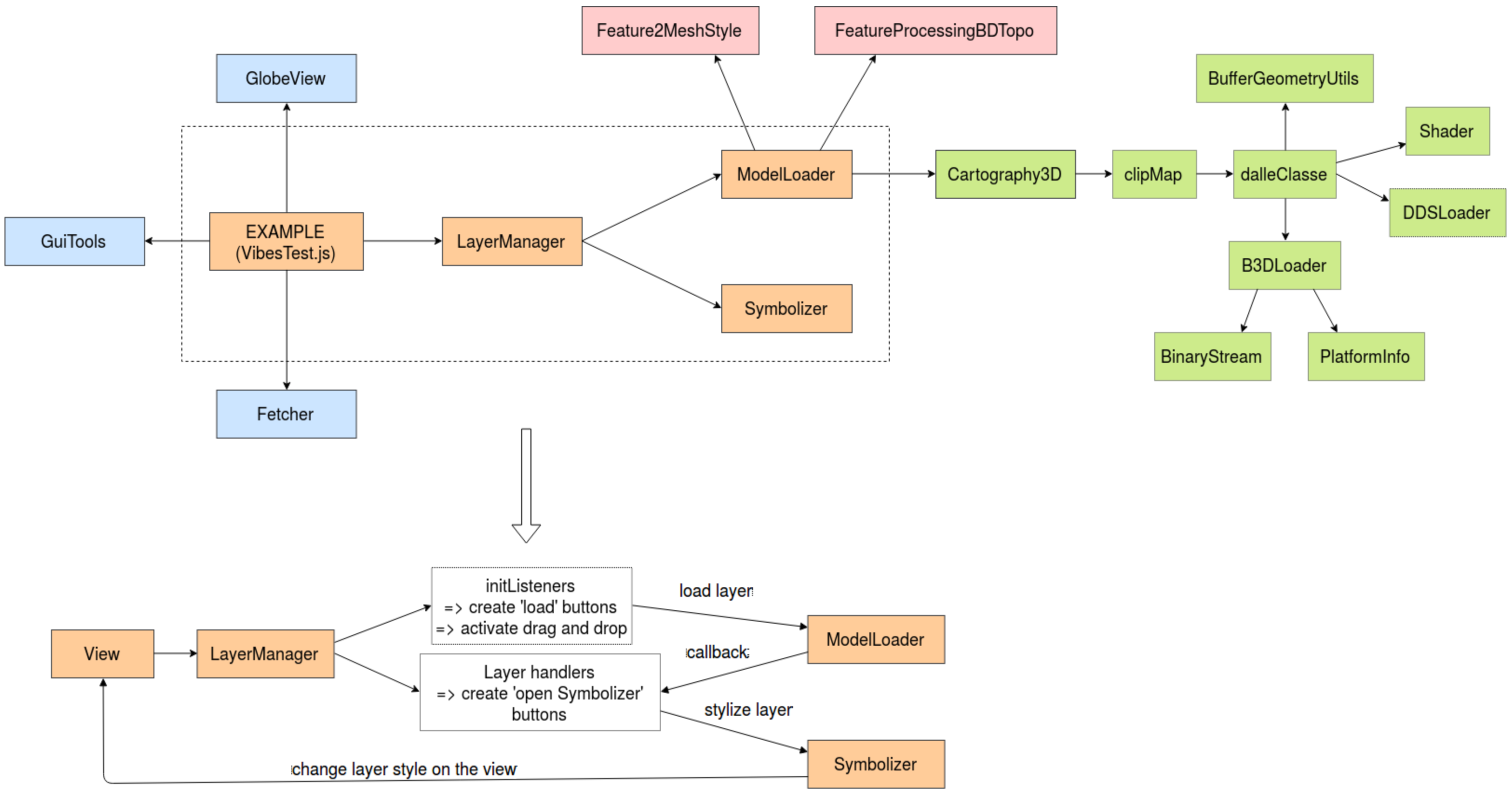


FIGURE 4: VIBES FINAL ARCHITECTURE

- The classes in orange are the ones we created from scratch.
- The classes in blue are the iTowns classes we re-used directly
- The classes in pink are iTowns classes we duplicated to make some slight modifications. (see **BD TOPO** for more details).
- The classes in green are classes from iTowns-legacy we re-used to load BATI3D (see **BATI3D** for more details).

CLASSES

The core of our project are the 3 classes and the example represented in orange:

- **ModelLoader.js**: the class to loads different sort of 3D objects (just *.OBJ* for now).
- **Symbolizer.js**: the class that carries all the stylization functionalities.
- **LayerManager.js**: the class that manages the user interface.
- **VibesTest.js**: the example file (linked to the HTML document) where we call the previous classes.

This schema describes the links between these classes:

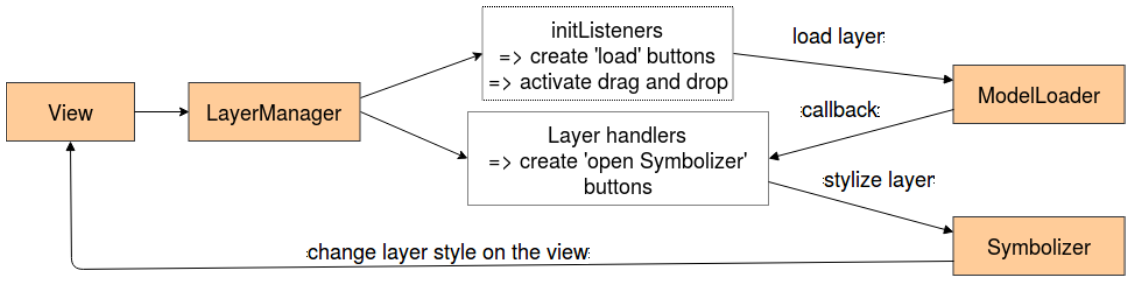


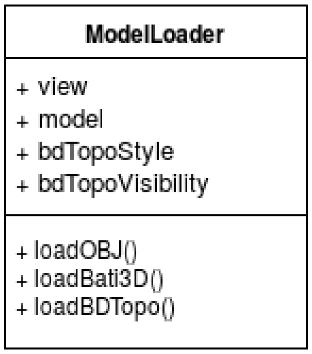
FIGURE 5: DETAIL OF THE OPERATION OF THE STYLIZATION PROCESS IN VIBES

- The LayerManager is instantiated in the example.
- The LayerManager initialize event listeners: 2 buttons on the GUI (to load BD Topo and BATI3D), and a drag and drop listener (to load OBJs).
- When one of these listeners is triggered, ModelLoader is called (*'load layer'* arrow).
- After the layer is loaded, it appears on the view and the LayerManager *handles* it (*'callback'* arrow). This means that the layer is added to a list of checkboxes, and buttons to activate the Symbolizer are created: the layer is ready to be stylized.
- When the user activates the Symbolizer (*'stylize layer'* arrow), the stylization controllers appear on the GUI, which allows the user to modify the visual aspect of the object.
- Stylization changes are displayed in the view.

CLASS MODELLOADER

This class has 2 attributes:

- **The iTowns view**
- **The object to load**: the model that carries the object loaded and the edges extracted from it (see EDGES EXTRACTION), and special attributes to handle BD Topo.

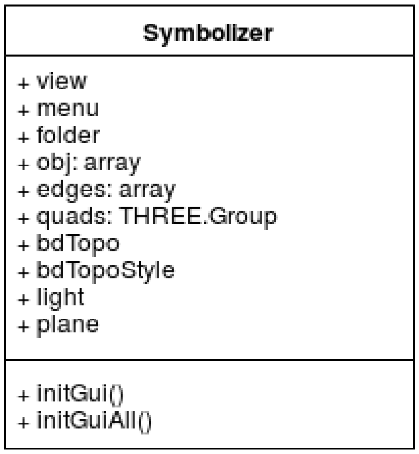


It contains one public method for each format: **loadOBJ()**, **loadBATI3D()**, and **loadBDTopo()**. These functions convert the 3D object into a group of meshes adapted to the symbolizer and call internal methods to load the object in iTowns. The final object (and its edges) are stored in the attribute *model*, except the tiles from BD Topo, which are handled differently.

CLASS SYMBOLIZER

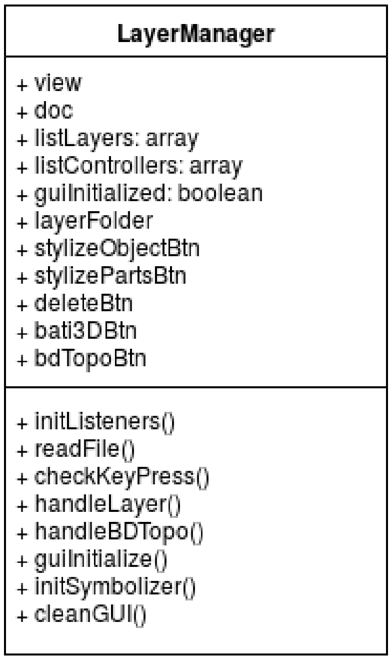
This class has the following attributes:

- **The iTowns view**
- **Attributes related to the GUI management**: the menu and the Symbolizer folder.
- **The objects to stylize**: the object itself (a list containing a group of *THREE. Mesh* for each layer), the edges, the possible quads (useful for the sketchy stylization), and special attributes for the stylization of the BD Topo extruded features.

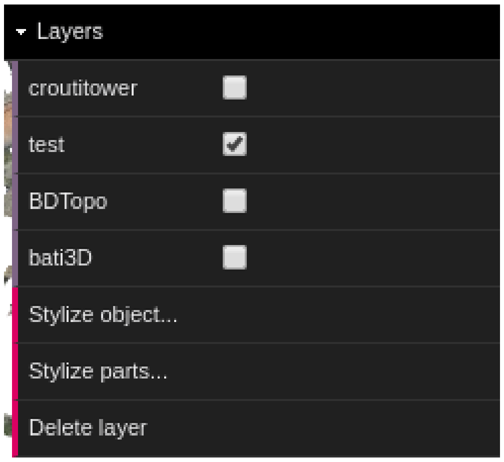


To initialize the Symbolizer, the user needs to call either initGui() or initGuiAll(). The operation of these methods is explained with more details in **GENERAL FUNCTIONING OF THE SYMBOLIZER**.

CLASS LAYERMANAGER



The ModelLoader and the Symbolizer could suffice to perform a stylization on an object. However, it is desirable to apply the same stylization on several objects. This is the interest of the LayerManager: providing an interface similar to those we can find in a GIS, so the user can manipulate his layers.



The methods of this class manage the elements of the GUI and the event listeners, as described in **USER INTERACTION WITH LAYERS**.

This class also allows to move the loaded object using check keys, as described in **GEOLOCATION**.

RESULTS

BASIC FUNCTIONALITIES

The first version of our tool was first based on the iTowns example "collada". It is located on a new example called "VibesObj". To try it, simply run this example on our fork of iTowns, available at **OUR GITHUB REPOSITORY**[4], or try it online **ON THE ITOWNS-RESEARCH REPOSITORY**[5].

LOADING A 3D OBJECT IN ITOWNS

The first step to stylize a 3D object is to load this object and make it visible. We focused on the .OBJ format at the beginning, as we already had samples for testing. We used the Three.js extension *OBJLoader*, already included in iTowns in the node module three-obj-loader (**SOURCE**[6]).

To load a 3D object in iTowns, we have to follow the following steps:

- Instantiate the globe.
- Instantiate the OBJLoader and call the load function.

The following steps are implemented in the callback of the load function.

- Place the object on its right location, rotate and scale it if necessary.
- Put the object layer in the camera layers so it is rendered.
- Initialize a material and assign it to the object.
- Update the transformation (with updateMatrixWorld()).
- Add the object to the scene.
- Notify the change to the globe view.

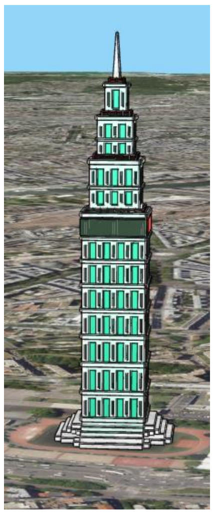


FIGURE 6: CROUTITOWER

The loaded object should now appear on the globe at the chosen position. We chose to first display it with a *THREE.MeshPhongMaterial*.

We implemented a drag and drop functionality to easily load the 3D object (on .obj format), with a fixed geolocation at first. The example models are located in examples/model. We have been using croutitower.obj (image right), test.obj and destroyer.obj for our first tests.

[4] https://github.com/arnaudgregoire/vibes

[5] https://github.com/itownsResearch/2018_TSI_vibes

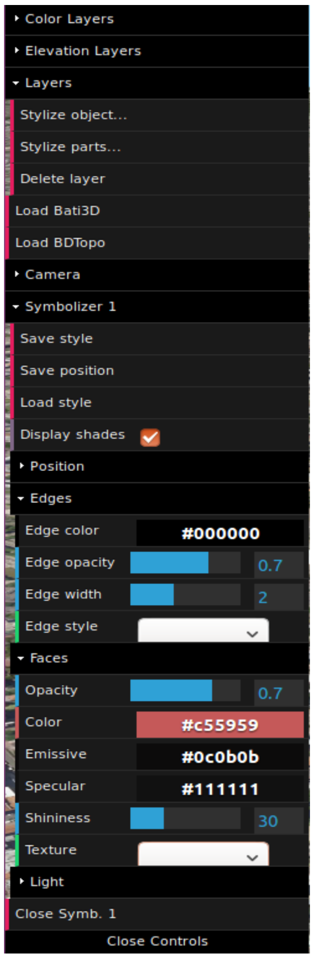
[6] https://github.com/sohamkamani/three-object-loader

APPLYING A STYLE TO A MESH WITH THREE.JS

To change the stylization of an object, we must know how this object is structured and where the information about its aspect is stored.

The objects we just loaded are actually a group of meshes (type *THREE.Group*). For example, the croutitower is composed of 14 meshes. We can access these meshes by iterating over the children of the object. Then we just have to access the attribute *material* of each mesh and change the attributes we want to change.

The basic implemented parameters are: **color**, **opacity**, **emissive color**, **specular color**, and **shininess**.



Display shades ✓
Edge color #000000
Edge opacity 0.7
Edge width 2
Edge style
Opacity 0.7
Color #c55959
Emissive #0c0b0b
Specular #111111
Shininess 30
Texture

FIGURE 7: USER INTERFACE

CREATING A USER INTERFACE TO DYNAMICALLY MODIFY THE STYLIZATION

The JavaScript library DAT.GUI[7] allows to create a user simple interface with buttons, sliders, checkboxes, etc. It is already used in iTowns, in the GuiTools class, to handle color and elevation layers on the globe. Thus, we will re-use this menu and add our own stylization parameters on it. Each element of the menu has an event listener with a callback function that performs the corresponding stylization on the mesh.

SAVING AND LOADING A STYLE

Our tool must also allow to save the current style in a *.vibes* file (see THE STYLE FORMAT) and re-load it later. We used at first a npm package function but after facing issues related to the CONTINUOUS INTEGRATION, we use a homemade function to save the file as a Blob object.

We used the JavaScript object *FileReader* to load a file and get the data in it. This data can then be parsed in JSON and read directly to be applied to the meshes.

When a stylesheet is loaded, the values of the GUI are updated to match the current stylization of the object.

[7] https://github.com/dataarts/dat.gui

ADVANCED FUNCTIONALITIES

LAYER MANAGEMENT

USER INTERACTION WITH LAYERS

As soon as the iTowns view is loaded, the user can add a layer to it. There is two ways to do this, depending on the format of the layer.

- An OBJ file can simply by dragged and dropped in the view.
- Layers from the databases BATI3D or BD Topo can be activated by clicking on the corresponding 'load' button on the GUI.

After the layer is loaded, its name appears on the GUI with a checkbox, so it can be selected (the user also can select an object of the view by clicking on it).

When one layer (or more) is checked, three buttons appear:

- **Stylize object**: open a symbolizer to stylize all the meshes of the objects at once.
- **Stylize parts**: open a symbolizer to stylize the meshes of the objects independently (the objects selected must have the same number of meshes).
- **Delete layers**: delete the objects.

These buttons disappear when there are no more layers checked (if they are all unchecked or deleted).

The following figure shows the sequence diagram for displaying the Symbolizer.

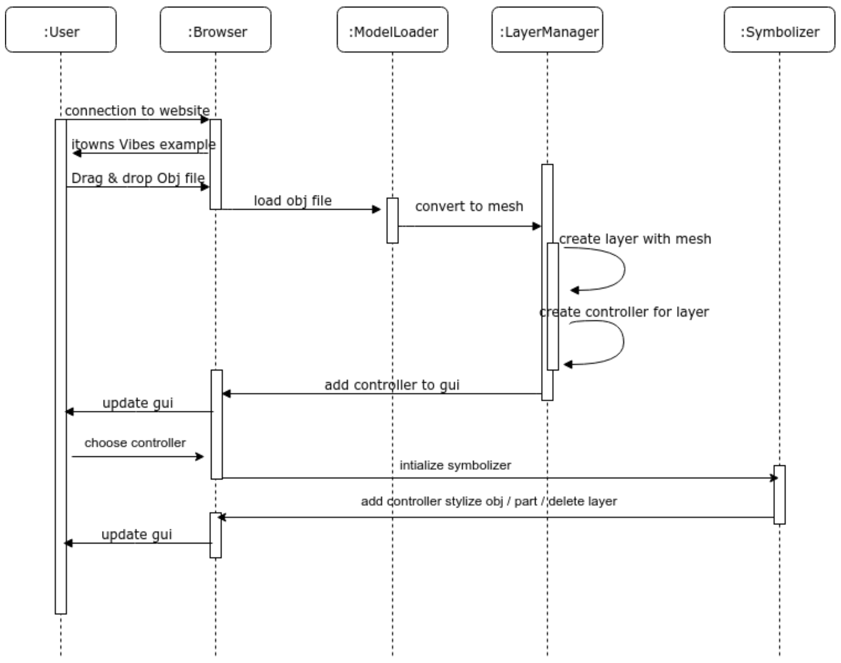


FIGURE 8: SEQUENCE DIAGRAM FOR VIEWING THE SYMBOLIZER.

The following figure shows us the user's interaction with the Symbolizer.

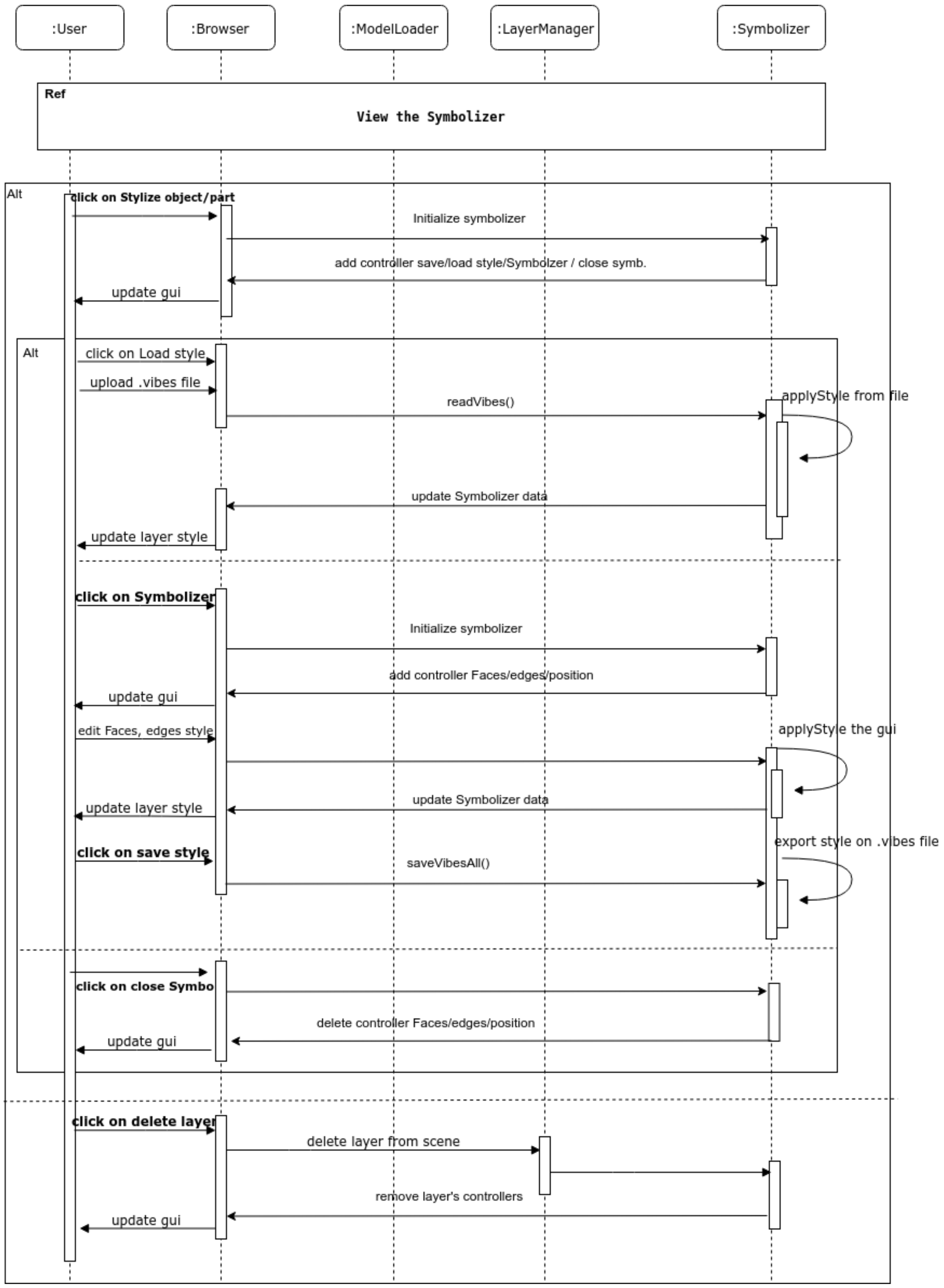


FIGURE 9: SEQUENCE DIAGRAM FOR INTERACTION WITH THE SYMBOLIZER.

GEOLOCATION

An important issue concerning the layers is how to **geolocalize** them. This is easy when the data itself is georeferenced, but formats like .OBJ do not provide this information. Therefore, in this case, the user should tell where the object is located, but the question is how.

The answer to this issue is twofold:

- The user should be able to enter (somehow) the parameters to locate the object he wants to stylize.
- He also should be able to adjust the position he chose (slight translations, rotations, scaling) later.

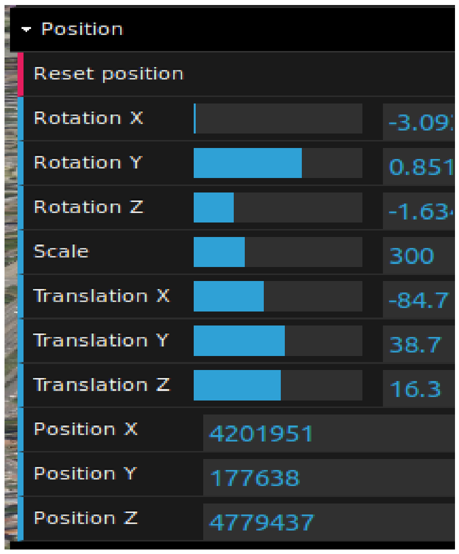
ADJUSTMENTS AND RELATIVE POSITIONNING

There are two ways to move objects:

The first one can be done by using the keyboard keys after clicking on the object or selecting it from the GUI, the user can use the following keys:

- Keys A and Z or 4 and 6 to move the object from West to East.
- Keys Q and S or 8 and 2 to move the object from North to South.
- Keys W and X or 7 and 3 to move the object from Top-Down.

The user also can use the sliders on the GUI which are shown in the picture right: Translate X, Translate Y and Translate Z


▾ Position

Reset position

Rotation X	-3.09
Rotation Y	0.851
Rotation Z	-1.63
Scale	300
Translation X	-84.7
Translation Y	38.7
Translation Z	16.3
Position X	4201951
Position Y	177638
Position Z	4779437

FIGURE 10: POSITION FOLDER OF THE GUI

ABSOLUTE POSITIONNING

But this method cannot be used to georeference an object completely - we cannot use a slider to move a mesh from one end of the world to the other. Until this step, the coordinates were hard-coded in the example, which is not satisfying.

A first solution is for the user to enter the coordinates directly in the GUI, and to adjust the position using the sliders. But this can be long and repetitive, especially if the user loads several objects at the same place.

Therefore, we went for a second solution, where the user can save a location file (format *.gibes*) containing the necessary parameters to put the object at its right position (coordinates, rotations and scaling). The file looks like the example right.

```
{
    "name": "croutitower",
    "coordX": 2.396159,
    "coordY": 48.848264,
    "coordZ": 50,
    "rotateX": 0.5,
    "rotateY": 0,
    "rotateZ": 0,
    "scale": 300
}
```

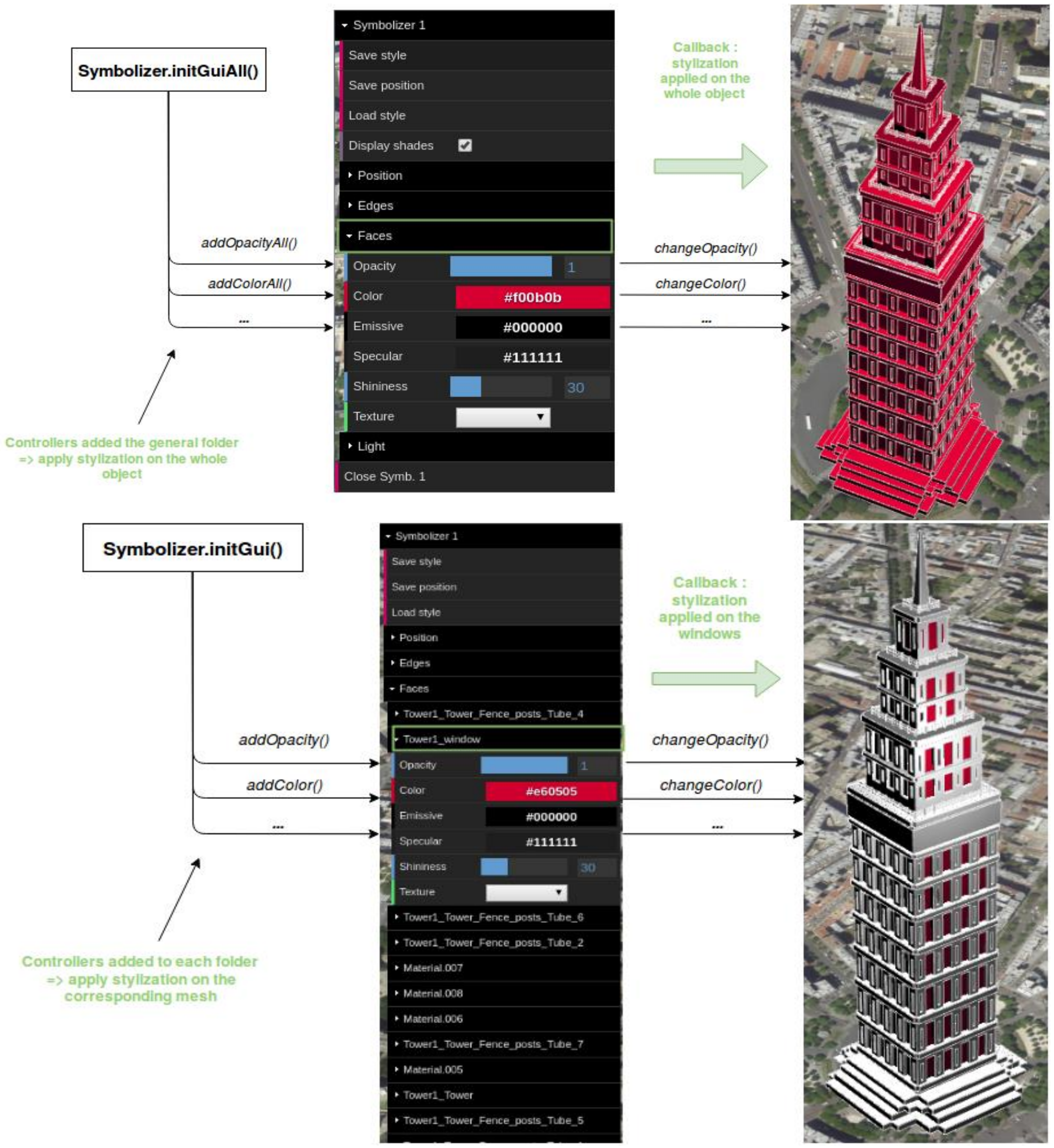
It can be dragged and dropped at any time and will be applied to all the checked layers in the GUI.

GENERAL FUNCTIONING OF THE SYMBOLIZER

The Symbolizer is the central class of Vibes, as it carries the concrete stylization functionalities. An object can be stylized in two ways: a global stylization, or a detailed stylization. In the second case, we stylize the object mesh by mesh, whereas in the first, we apply the same style everywhere. Therefore, there is two methods to initialize the Symbolizer:

- **initGuiAll**: opens one Symbolizer for all the meshes of the object.
- **initGui**: opens one Symbolizer for each mesh.

The process of stylization in the Symbolizer works as follows:


Symbolizer.initGuiAll()
addOpacityAll()
addColorAll()
...
Controllers added the general folder => apply stylization on the whole object
Symbolizer 1
Save style
Save position
Load style
Display shades
Position
Edges
Faces
Light
Close Symb. 1
Callback : stylization applied on the whole object
changeOpacity()
changeColor()
...
Symbolizer.initGui()
addOpacity()
addColor()
...
Controllers added to each folder => apply stylization on the corresponding mesh
Symbolizer 1
Save style
Save position
Load style
Position
Edges
Faces
Tower1_Tower_Fence_posts_Tube_4
Tower1_window
Tower1_Tower_Fence_posts_Tube_6
Tower1_Tower_Fence_posts_Tube_2
Material.007
Material.008
Material.006
Tower1_Tower_Fence_posts_Tube_7
Material.005
Tower1_Tower
Tower1_Tower_Fence_posts_Tube_5
Callback : stylization applied on the windows
changeOpacity()
changeColor()
...

Opacity 1
Color #f00b0b
Emissive #000000
Specular #111111
Shininess 30
Texture

Opacity 1
Color #e60505
Emissive #000000
Specular #111111
Shininess 30
Texture

FIGURE 11: GENERAL FUNCTIONING OF THE TWO SYMBOLIZERS

Each initializer method builds the structure of the GUI, with the appropriate folders and add the controllers to it (buttons and sliders). These controllers all carry callback functions that perform the concrete stylization on the object or edges when they are triggered.

EDGE STYLIZATION

EDGE EXTRACTION

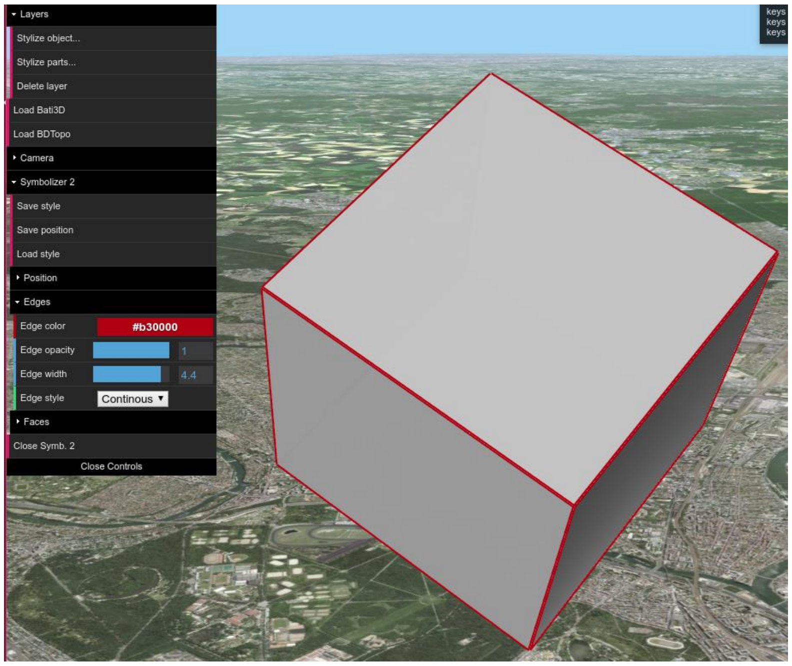
The edges are extracted from the geometry thanks to a *THREE.EdgesGeometry* object, then converted into *THREE.LineSegments* and added to a group of lines that will be placed in the scene at the same coordinates as the object.

These edges are initialized with a *THREE.LineBasicMaterial* that can be stylized the same way as the materials on the faces. However, unlike the faces, the edges can only be stylized as a whole, we did not separate them according to the mesh from where they were extracted.

To be able to stylize the edges we have the basics parameters such as : color, opacity, width, and we can texture the edges with different texture (continuous ,dashed, sketchy).

CONTINUOUS EDGES

The "Continuous" texture uses the basic texture of the materials in three.js, that is edges in the form of a continuous line.



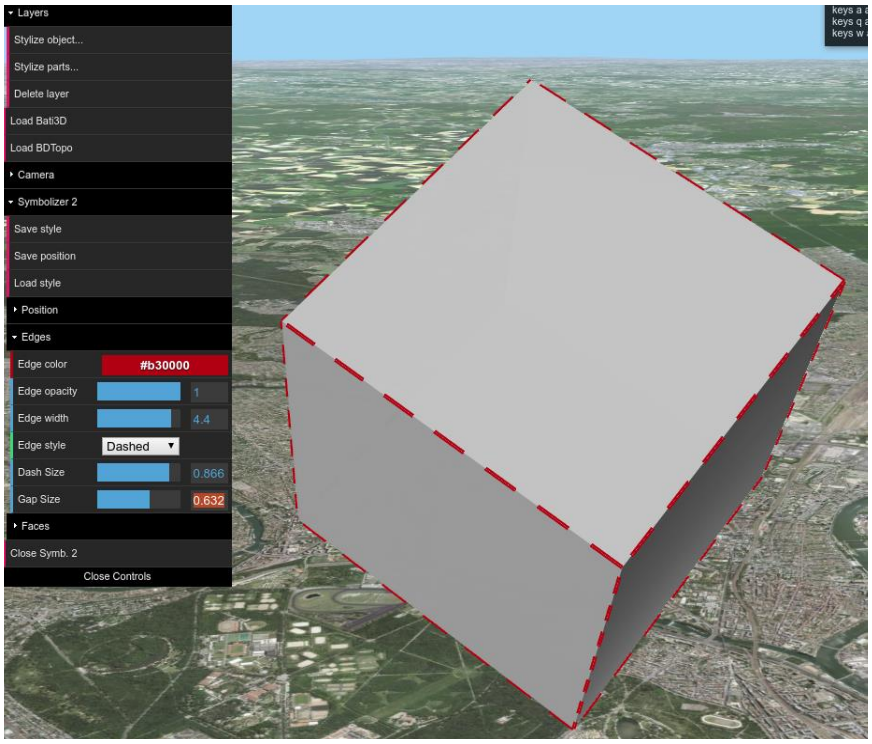
Edge color #b30000
Edge opacity 1
Edge width 4.4
Edge style Continous

FIGURE 12: *"CONTINUOUS" TEXTURE FOR EDGES*

DASHED EDGES

The "dashed" texture sets the edges as a dotted line and this choice triggers the addition of two parameters : dash size and gap size.

To achieve the "Dashed" edges we encountered a problem with the Three.js library. Indeed, this parameter requires a function of Three.js that has been moved into class *THREE.Line* in a version later than the one included in iTowns, and we could not operate this function at its previous location. To solve the problem, we had to update the version of three.js in iTowns.



Edge color #b30000
Edge opacity 1
Edge width 4.4
Edge style Dashed
Dash Size 0.866
Gap Size 0.632

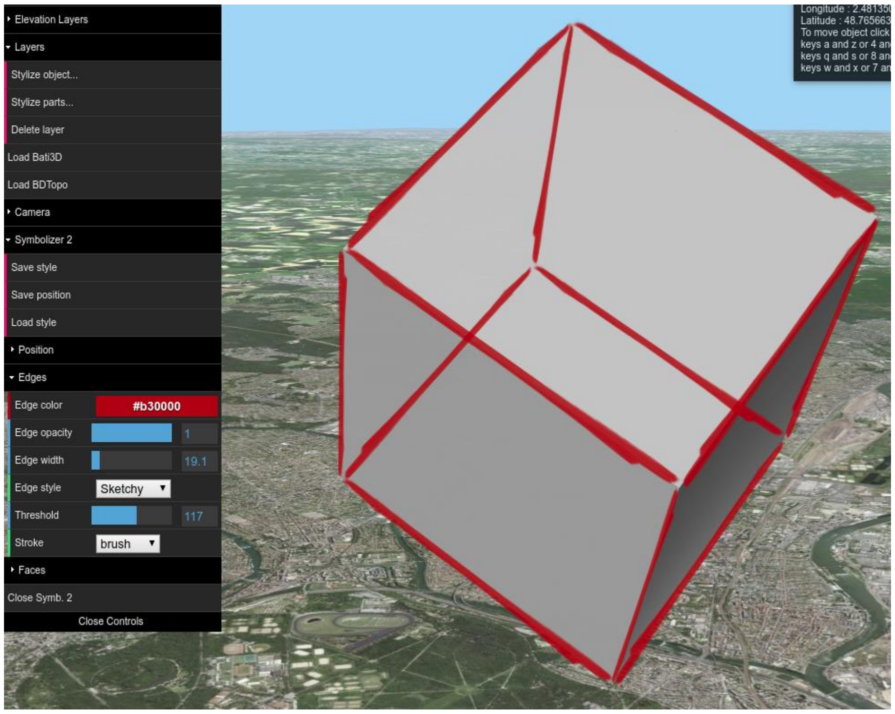
FIGURE 13: *"DASHED" TEXTURE FOR EDGES*

SKETCHY EDGES

The “Sketchy” texture sets the edges to look line a drawn line, which give them a sketchy aspect, similar to a cartoon. To do that, we must apply a texture that represents a brushstroke. However, an edge is a linear geometry, so we cannot simply apply a texture on it. A solution, based on Mathieu Bredif's work, has already been found in the PLU++ project. It consists in creating a quadrilateral where the edge is located and apply a specific shader to it. This rectangle should always be facing the camera, so the edge is always visible.

Setting the edge style to ‘sketchy’ adds two other parameters to the GUI : ‘stroke’ to choose the shape of drawing, and ‘threshold’ to determine what will be the threshold edge size under which we will apply a specific texture for small edges.

During the implementation of the ‘sketchy’ stylization, we faced problems related to the depth test in iTowns, so we bypassed it in the shader, but it causes visualization problems.



Edge color #b30000
Edge opacity 1
Edge width 19.1
Edge style Sketchy
Threshold 117
Stroke brush

Longitude : 2.481350
Latitude : 48.765663
To move object click
keys a and z or 4 an
keys q and s or 8 an
keys w and x or 7 an

FIGURE 14: *"SKETCHY" TEXTURE FOR EDGES.*

FACE STYLIZATION

SIMPLE PARAMETERS

When the 3D object is loaded, it is converted to meshes. The material of these meshes are initialized with a *THREE.MeshPhongMaterial* object so they can be stylized using a Symbolizer. The parameters which can be currently change are: **opacity**, **color**, **emissive**, **specular**, **shininess**.

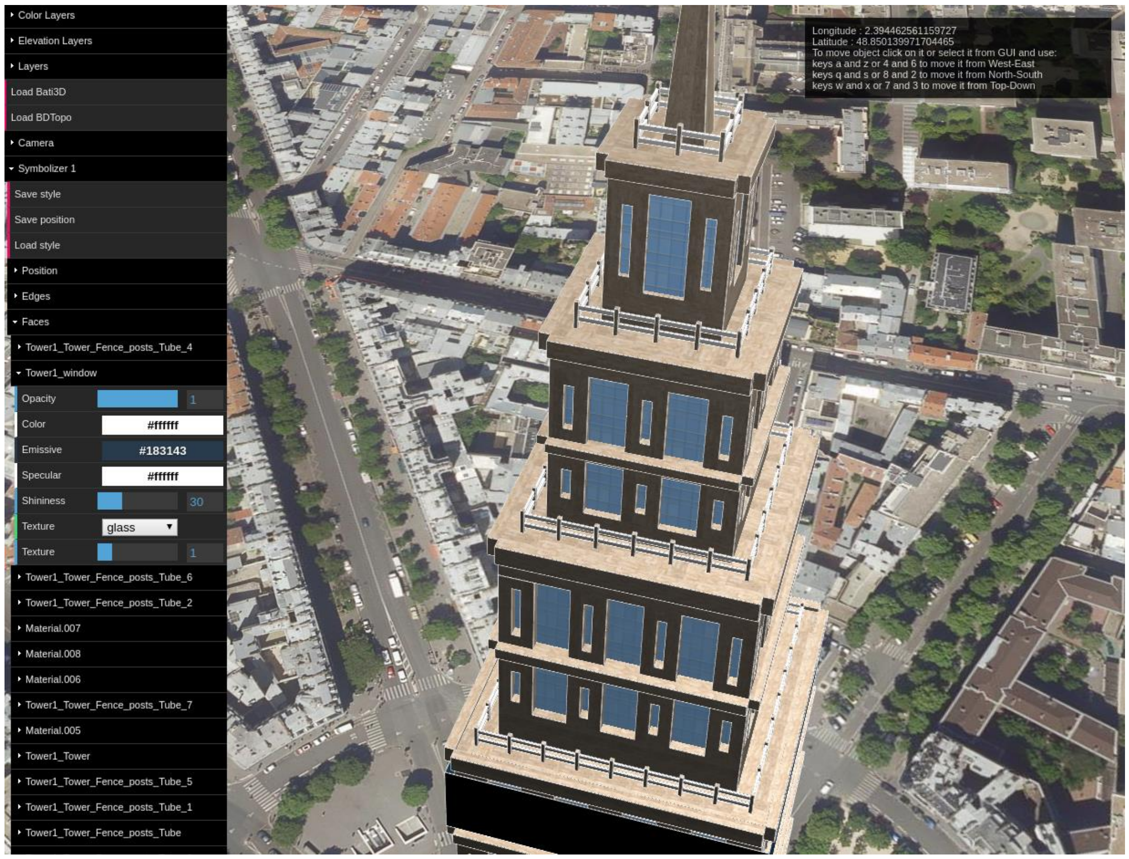
FACE TEXTURATION

The PLU++ project allows to apply texture on the faces of the object in order to diversify the possible styles. The images we used as sample textures were taken from this project and from the croutitower example.

Applying a texture on a face is rather easy: the *THREE.MeshPhongMaterial* has a 'map' parameter that can store a texture. We used *THREE.TextureLoader* to load an image from its local path and added the texture we obtained to the material.

The source image must be located in the right folder in iTowns (*examples/textures*) and the name of the texture must appear in the *listeTexture.json*. The path of the texture is saved in the stylesheet.

When a texture is applied, a new slider appears on the GUI to change the repetition of the texture.


Color Layers
Elevation Layers
Layers
Load Bati3D
Load BDTopo
Camera
Symbolizer 1
Save style
Save position
Load style
Position
Edges
Faces
Tower1_Tower_Fence_posts_Tube_4
Tower1_window
Tower1_Tower_Fence_posts_Tube_6
Tower1_Tower_Fence_posts_Tube_2
Material.007
Material.008
Material.006
Tower1_Tower_Fence_posts_Tube_7
Material.005
Tower1_Tower
Tower1_Tower_Fence_posts_Tube_5
Tower1_Tower_Fence_posts_Tube_1
Tower1_Tower_Fence_posts_Tube

Opacity	1
Color	#ffffff
Emissive	#183143
Specular	#ffffff
Shininess	30
Texture	glass
Texture	1

Longitude : 2.394462561159727
Latitude : 48.850139971704465
To move object click on it or select it from GUI and use:
keys a and z or 4 and 6 to move it from West-East
keys q and s or 8 and 2 to move it from North-South
keys w and x or 7 and 3 to move it from Top-Down

FIGURE 15: CROUTITOWER WITH 'BETON' AND 'GLASS' TEXTURES

SHADER APPLICATION

To allow the user to apply a more customized render, a next step could be the application of a shader in the faces of an object. This could be done the same way as we did to create sketchy edges, with the *THREE.ShaderMaterial*.

Like for the texturation with the image, some default shaders would be located in a folder in iTowns, with a json file containing the lists of names, in order to make them appear in the GUI as a drop-down list. Then the user would be able to add its owns shaders.

For each shader, three files would be required:

- The **vertex** shader: *ShaderName_vert.glsl*
- The **fragment** shader: *ShaderName_frag.glsl*
- A JSON file containing the **uniforms**: *ShaderName_uni.json*

This functionality might be implemented in the last week of the project.

ENVIRONMENT

Customizing the stylization of the environment in iTowns is a little more challenging than the other parameters, as it implies acting on elements that are already implemented. Unlike PLU++, the environment is already set, so we cannot re-use the functions.

LIGHTS

In order to make the integration of a 3D mesh more realistic, we had a light located nearly above the building. It is a *THREE.PointLight* that gets emitted from a single point in all directions. We would have preferred to use a *THREE.DirectionalLight* like the one already implemented internally in iTowns, but we never achieved to make it work. The drawback of *PointLight* is that two identical objects in the 3D scene located at different places will have different shadows.

```
var plight = new THREE.PointLight(0xffffff, 1, 0, 1);
var coordLight = coord.clone(); // Building coordinates
coordLight.setAltitude(coordLight.altitude() + 350);
plight.position.copy(coordLight.as(this.view.referenceCrs).xyz());
plight.position.y += 70;
```

Moreover, we had in the dat.GUI a subfolder named "Light" which contain sliders that are linked to light position (x, y, z) and color. For example, the user turns the light to yellow and can move it to see shadow rotation. This can be a first approximation of a daylight.

SHADOWS

In a native iTowns application, there is no easy way to implement buildings shadow. In fact, iTowns modify the classic shadow mapping method, making it not usable for us. In order to create shadows of buildings on the ground, we create 2 objects:

- a 1x1km plan centered on building. This plan is made of *THREE.ShadowMaterial*. This particular three.js material can receive shadows, but otherwise is completely transparent. We make it semi-transparent in order to have a nice shadow (less dark). Unfortunately, we had to disable the *depthTest* to make our shadow appear. According to our product owner Alexandre Devaux, its due to a bug internally to iTowns. This lack of *depthTest* makes sometimes a visual bug where shadows will be put in front of the building.

```
var planeGeometry = new THREE.PlaneBufferGeometry(20, 20, 32, 32);
var planeMaterial = new THREE.ShadowMaterial({ side: THREE.DoubleSide, depthTest:
false });
planeMaterial.transparent = true;
planeMaterial.opacity = 0.5;
var plane = new THREE.Mesh(planeGeometry, planeMaterial);
```

- The light described above.

Finally, we add an option "Display shades" in our dat.GUI to let to the user the choice of displaying our shadows. Here is a screenshot of one building with it shadow.

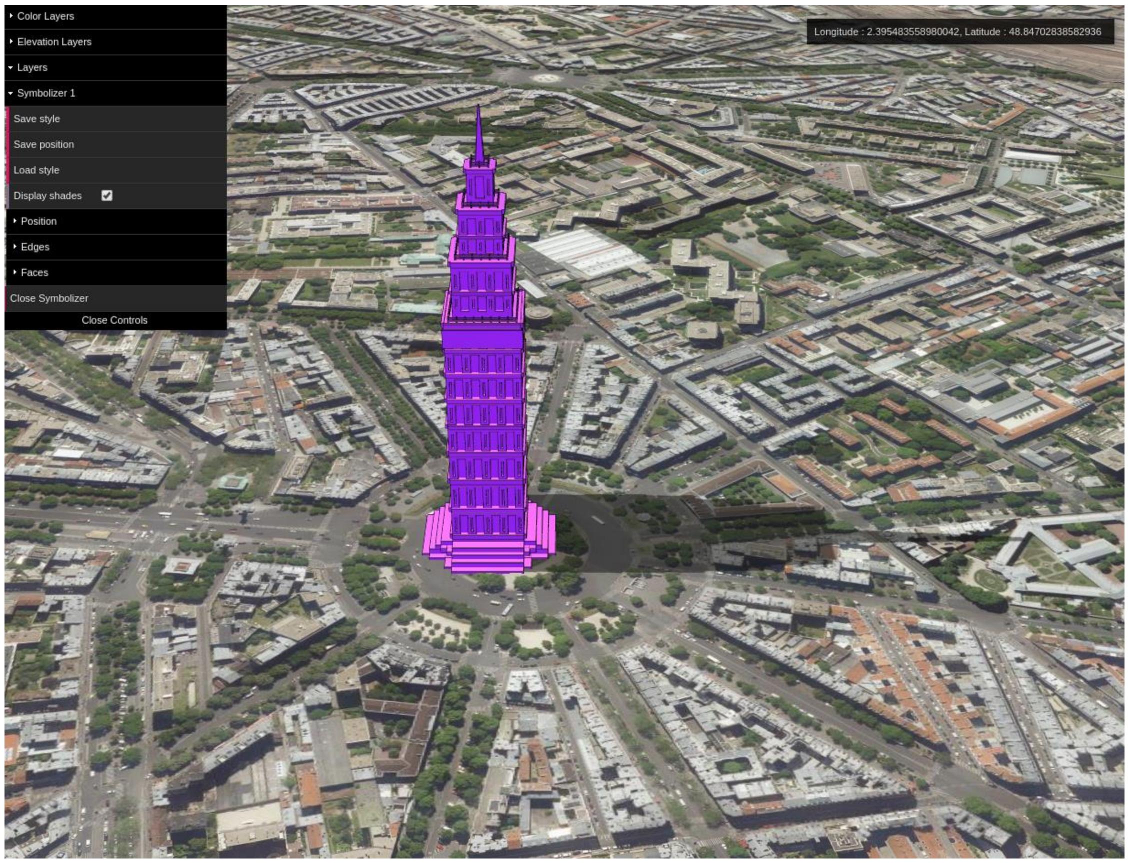


FIGURE 16: CROUTITOWER WITH SHADOW

CAMERA

Once the question of geolocation was resolved, we had to manage the camera orientation. In iTowns, some function can help to manage the camera. For instance, to follow the movement of the 3D object on the scene, a function is called when its coordinates change:

```
globeView.controls.setCameraTargetPosition(this.obj[0].position, false);
```

The same function is used when the object is load on the scene. The position of the object becomes the position of the camera. This system is managed in the class Symbolizer.

Also, it is possible to modify the coordinates of the camera and the zoom scale (parameters **Longitude**, **Latitude** and **Zoom**) and reinitialize the parameters with the menu.

In addition, different camera points of view are proposed: **oblique**, **immersive**, **globe**.

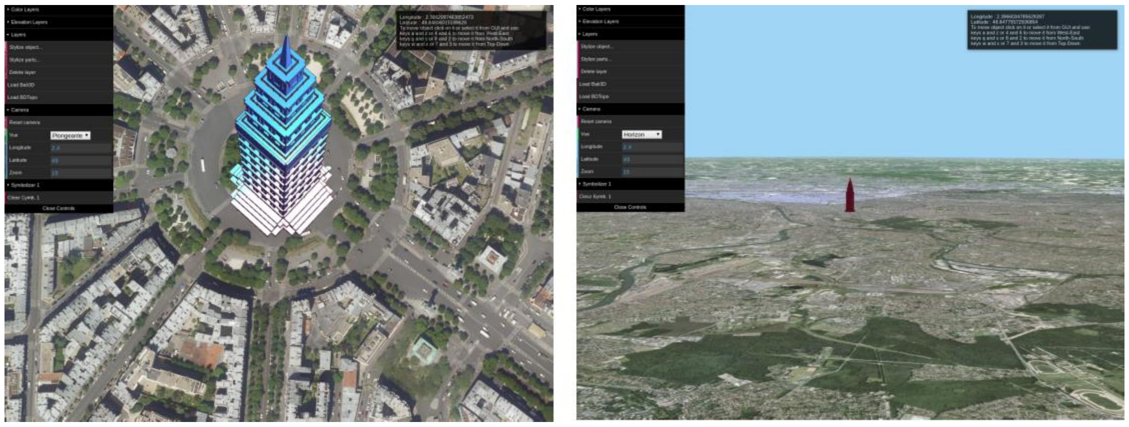


FIGURE 17: TWO POSSIBLE CAMERA ANGLES : 'BIRD'S EYE VIEW' AND 'HORIZON'

LOADERS

We have three different loaders for loads three types of object (.obj, BATI3D and BDTopo).

OBJ LOADER

The *'.obj'* is a standard to record 3D objects. *three.js* already have a loader to convert *'.obj'* to *THREE.Group*. We used the npm package *'three-obj-loader'* to load the *'.obj'* and we created a function to personalize them. So, we put it at the given location, apply the given rotation and scaling, create and apply the initial THREE.MeshPhongMaterial on the faces and extract the edges and initialize their style. Then, we manage the light and the shadow for the group. Finally, we add the light, the shadow, the faces and the edges to the scene and after use the giving callback function with the faces and the edges.

BATI3D LOADER

The BATI3D is an IGN production that provides the 3D building of France in 500mx500m tile (in the localization where the data exists). A tile match a folder whose name depend on the top left corner coordinates (ex:'EXPORT_1302-13722'). The 3D model is saved as '.3DS' file and linked to the corresponding orthophoto images. Each point of the 3D model is geolocated with *Lambert93* coordinates (EPSG:2154).

To load BATI3D data, we followed the process used in the IGN project: ***ITOWNS-LEGACY***[8] that loads a sample of BATI3D in an iTowns plan view. The difficulty is to make the load work on the iTowns globe view instead of the iTowns plan view. Another difficulty is the points coordinates which is expressed in *Lambert93* whereas iTowns only uses the Geocentric coordinate system *WGS84* (EPSG:4978) and the Geodetic coordinate system *WSG84* (EPSG:4326).

[8] https://github.com/iTowns/itowns-legacy

We reused the classes: **Cartography3D**, **clipMap**, **dalleClasse**, **Shader**, **B3DLoader**, **BinaryStream**, **DDSLoader**, **PlatformInfo** and the function **BufferGeometryUtils** extracted from **Utils**.

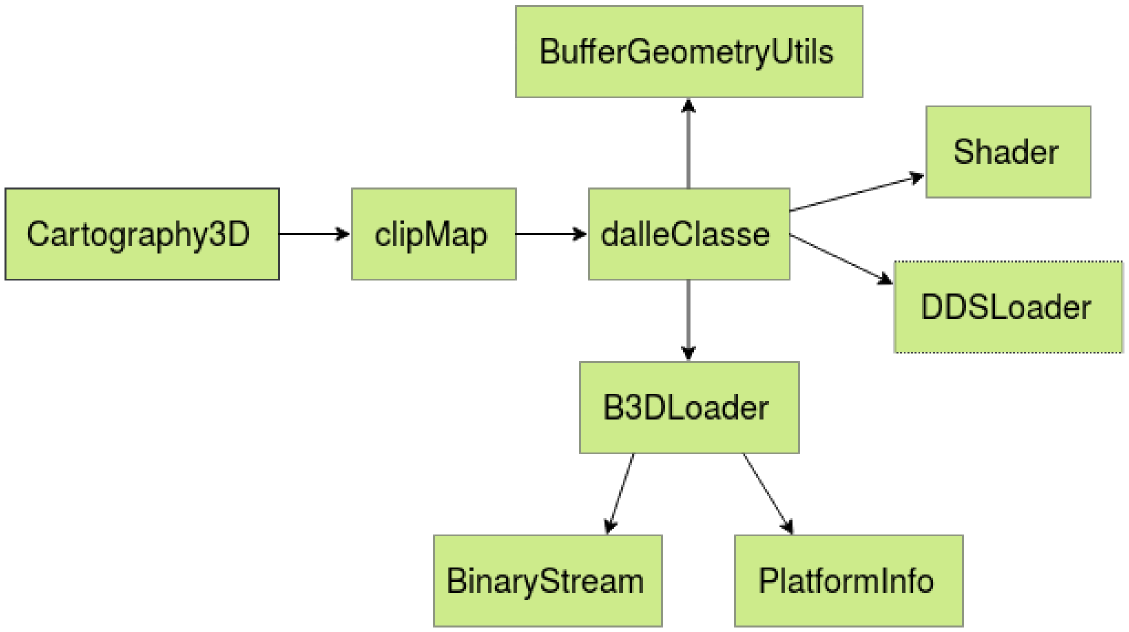


FIGURE 18: ARCHITECTURE OF THE CLASSES TO LOAD BATI3D IN ITOWNS

- **Cartography3D** initializes the creation of the BATI3D object. We change the refocusing of the tile, the way of loading the tile which initially depended on the camera position and now depends only on the area and the available tile.
- **clipMap** creates the grid of all **dalleClasse** use.
- **dalleClasse** (tileClass) loads the BATI3D data with the **B3DLoader**, creates the *THREE.Group* of faces with **BufferGeometryUtils**, extracts the group of edges, applies the right material and adds them on the scene.
- **B3DLoader** reads the binary BATI3D data and extracts all the information needed.
- **BinaryStream**, **PlatformInfo** are used by **B3DLoader** to read the binary BATI3D data.
- **DSLoader** is used to read the orthophoto images (not used now).
- **BufferGeometryUtils** converts the information extracted by the *B3DLoader* to a *THREE.Group* of *THREE.BufferGeometry*. We changed the way of creating the *THREE.BufferGeometry*: the coordinates (EPSG:2154) are converted to EPSG:4978, put in the order (x, y, z) and not (x, z, y) and we add the 'normal' so the *THREE.MeshPhongMaterial* can be applied right. We use a *THREE.MeshPhongMaterial* for the faces and create the edges associated.

We had some problems with the orthophoto images application, so we did not use it.

BD TOPO® LOADER

The BD TOPO® is a *"3D vector description (structured in objects) of the elements of the territory and its infrastructures, of metric precision, exploitable on scales ranging from 1: 5 000 to 1: 50 000."* **SOURCE**[9]. The BD TOPO® is accessible with a WFS flux and iTowns already use it on the **GLOBE WFS EXTRUDED**[10] to load the buildings. We do the same as the iTowns example, but we need to change the way to create the visible object and manage the layer.

We created a new class Feature2MeshStyle based on Feature2Mesh. We changed the creation of the object, the roofs and the walls are separated, the edges are created and a THREE.MeshPhongMaterial is applied instead of a vertexColor. The parameters of the faces and edges material depend on the style updated by the *symbolizer*.

During the symbolization of the BD TOPO® we had some problems with the opacity of the walls, roofs and edges which is not applied to the buildings. The problem came from the update function *'FeatureProcessing.update'* which makes the mesh opacity equal to the Layer opacity. So, we created *'FeatureProcessingBDTopo.update'* which does not change the customized parameters of the mesh.

We use a flux to get the BD TOPO® data so it is not handled like the other objects so we created a function, *'ForBuildings'*, on the *ModelLoader* to access at the Mesh of each BD TOPO®'s tiles and can edit them.

[9] http://professionnels.ign.fr/bdtopo

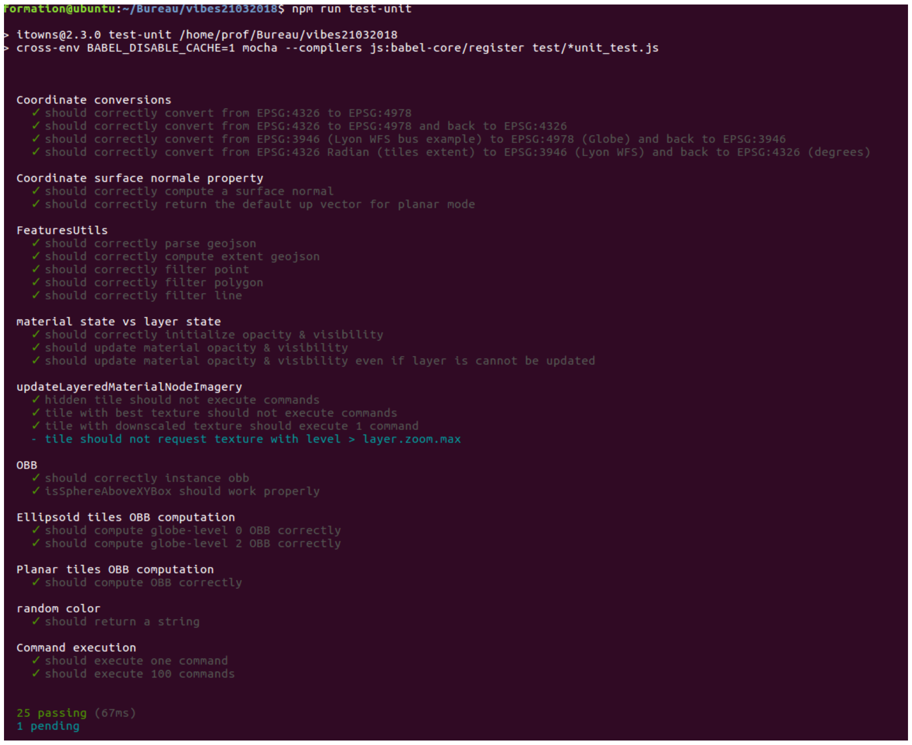
[10] http://www.itowns-project.org/itowns/examples/globe_wfs_extruded.html

TESTS

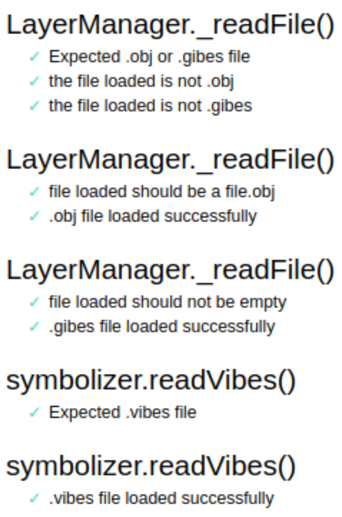
Unit tests

In order to write our unit tests, we rely on the mocha framework which was used in the previous unit tests of iTowns. This framework is a feature-rich JavaScript test that can be used for both Node.js and browser-based testing. Its interfaces system such as BDD, TDD, Exports, QUnit and Require-style allows developers to choose their style of DSL.

We added some tests to the previous test folder of iTowns. Some of these tests run using the CLI (commande line interface) and the others need the browser to be executed since our application uses the DOM element, and the node.js server does not have access to the DOM. Therefore, we had to run this kind of tests on the browser. To do that we made a simple HTML page (/browsermochaTest.html). The page loads Mocha, the testing libraries and our test file(/test/browsertest.js) and finally to run the tests, we simply needed to open the runner in a browser.



```
formation@ubuntu:~/Bureau/vibes21032018$ npm run test-unit

> itowns@2.3.0 test-unit /home/prof/Bureau/vibes21032018
> cross-env BABEL_DISABLE_CACHE=1 mocha --compilers js:babel-core/register test/*unit_test.js



  Coordinate conversions
    ✓ should correctly convert from EPSG:4326 to EPSG:4978
    ✓ should correctly convert from EPSG:4326 to EPSG:4978 and back to EPSG:4326
    ✓ should correctly convert from EPSG:3946 (Lyon WFS bus example) to EPSG:4978 (Globe) and back to EPSG:3946
    ✓ should correctly convert from EPSG:4326 Radian (tiles extent) to EPSG:3946 (Lyon WFS) and back to EPSG:4326 (degrees)

  Coordinate surface normale property
    ✓ should correctly compute a surface normal
    ✓ should correctly return the default up vector for planar mode

  FeaturesUtils
    ✓ should correctly parse geojson
    ✓ should correctly compute extent geojson
    ✓ should correctly filter point
    ✓ should correctly filter polygon
    ✓ should correctly filter line

  material state vs layer state
    ✓ should correctly initialize opacity & visibility
    ✓ should update material opacity & visibility
    ✓ should update material opacity & visibility even if layer is cannot be updated

  updateLayeredMaterialNodeImagery
    ✓ hidden tile should not execute commands
    ✓ tile with best texture should not execute commands
    ✓ tile with downscaled texture should execute 1 command
    - tile should not request texture with level > layer.zoom.max

  OBB
    ✓ should correctly instance obb
    ✓ isSphereAboveXYBox should work properly

  Ellipsoid tiles OBB computation
    ✓ should compute globe-level 0 OBB correctly
    ✓ should compute globe-level 2 OBB correctly

  Planar tiles OBB computation
    ✓ should compute OBB correctly

  random color
    ✓ should return a string

  Command execution
    ✓ should execute one command
    ✓ should execute 100 commands


  25 passing (67ms)
  1 pending
```



```
LayerManager._readFile()
  ✓ Expected .obj or .gibes file
  ✓ the file loaded is not .obj
  ✓ the file loaded is not .gibes

LayerManager._readFile()
  ✓ file loaded should be a file.obj
  ✓ .obj file loaded successfully

LayerManager._readFile()
  ✓ file loaded should not be empty
  ✓ .gibes file loaded successfully

symbolizer.readVibes()
  ✓ Expected .vibes file

symbolizer.readVibes()
  ✓ .vibes file loaded successfully
```

FIGURE 19: RESULTS OF THE UNIT TESTS RUN IN CONSOLE (LEFT) AND IN BROWSER (RIGHT)

Functional Tests

During the development we created a test protocol to make sure the develop functionality works well. When we added a new functionality, we updated this protocol. THE PROTOCOL IS ACCESSIBLE IN ANNEX 3.

Continuous Integration

To run all the tests, we use **TravisCI**[11]. The initial iTowns project already uses this tool, so we change the existing travis.yml to do only the tests without the deployment. This CI tool is rather strict : it does not allow any warnings messages in the compilation, and the code must follow the **ESLint** guidelines.

The CI failed after using the saving functionality due to problems with the npm package ***'file-saver'***[12], during the compilation of iTowns on 'itowns-testing.js'. The same problem happened with the package ***'savery'***[13]. Therefore, we used another save function written on the example and passed in the *Symbolizer*.

Deployment

In this project we worked on the repository **https://github.com/arnaudgregoire/vibes**. Mathieu Bredif wanted to gather iTowns examples on the repository **https://github.com/itownsResearch**. This is why we created a second repository **https://github.com/itownsResearch/2018_TSI_vibes** which host the online version of vibes at **https://itownsresearch.github.io/2018_TSI_vibes/examples/vibesObj.html**.

In order to publish on our website a given version of vibes, we developed a publish script (./publish.sh at root folder) that follow the workflow described below:

- First, we force push the new version of arnaudgregoire/vibes on itownsResearch. By doing that, all files from itownsResearch/2018_TSI_vibes are replaced by arnaudgregoire/vibes files.

```
git push -f https://github.com/itownsResearch/2018_TSI_vibes master:master
```

- Then we clone a version of the itownsResearch repository. We install all the dependency and we build the dist folder which contain all built JavaScript files. We will need those file in order to maintain our website without any node server, just pure html5.

```
git clone https://github.com/itownsResearch/2018_TSI_vibes
cd 2018_TSI_vibes
npm install
npm run build
```

Once we built our application, we commit and push the js files contained in the dist folder. This way, this preserve a readable tree of commit, preventing thousand lines of differences between JavaScript built files.

```
git add -f dist/*.js
git commit -m “dist”
git push -f
```

[11] https://travis-ci.org/arnaudgregoire/vibes

[12] https://www.npmjs.com/package/file-saver

[13] https://www.npmjs.com/package/savery

CONCLUSION

GENERAL REVIEW

For 6 weeks, we have worked together with real efficiency. We achieved in all goals that were asked by our backer Sidonie Christophe. We left behind us a fully functional and documented repository (HTTPS://GITHUB.COM/ITOWNSRESEARCH/2018_TSI_VIBES) and an online demo at this address : HTTPS://ITOWNSRESEARCH.GITHUB.IO/2018_TSI_VIBES/EXAMPLES/VIBESOBJ.HTML. We used a SCRUM method of development using most commonly used tools in the professional world like Trello, GitHub or Travis. Moreover, we used all the knowledge learnt this year, particularly in the fields of continuous integration and continuous deployment, design patterns, team management, node and JavaScript front-end and back-end, 3D visualization and above all iTowns. That leads us today to thank all professors that oversees our year, making it great and giving us a real technical background for our future work.

LIMITS AND PERSPECTIVES

Our team worked together during this project in order to achieve the set objectives, we succeeded in implementing various functionalities and techniques of buildings stylization, and we almost reached all the goals we had to complete. However, we faced some difficulties when developing our project. We can summarize them in the following points :

- Although we managed to display the shadows of our objects, we had to bypass the depth test in iTowns, which implies visual imperfections.
- We had a similar problem with the implementation of the sketchy stylization : the render can be improved.
- The majority of the unit tests that we intended to run them on node.js need to access the Dom element, but the node.js server does not have access to it, so we ran them on the browser. As we have yet to find a way to run these tests on a console, we are not able to include them in the CI. Therefore, they have to be run manually.
- We have not implemented the application of shaders on the faces of an object yet.
- Our tool does not allow to load the MTL file that goes along with the OBJ file yet.

Annexes

PERSONAL REVIEWS

HOUSSEM ADOUNI

As a developer I contributed in many parts of the project. for example : relative positioning and adjusting objects positions in order to give the user two ways to move the selected object (the keyboard keys or GUI) and I also contributed in edges style and edges texture, in addition I contributed on the layer management (see more detail in Trello). I worked most of the time with M. El Hadi, and also Laurie, Rose and Ludivine.

EL-HADI BOUCHAOUR

As a simple developer , I worked with the project team to achieve the set objectives, so I participated in developing of some functionalities which were assigned to me during the sprints. I started working together with Houssem on the edge style feature and I contributed in particular on how to manage the edge width, after that I worked on the unit tests with Adel and since we based on the iTowns project we found already some of them in the test folder, so we decided to continue working and trying to add more tests to this folder. Moreover, I worked with Houssem on the part of adjustments and relative positioning of the objects in order to allow the user to move the selected objects using keyboard keys or GUI sliders.

ARNAUD GRÉGOIRE

As a simple developer, I worked on three.js mesh integration in iTowns. Moreover, I worked on shadows and lights in iTowns Scene. Furthermore, I made the drag and drop and all interactions between computer system files and browsers. In addition to that, I made a set of examples. Besides, I although make the continuous deployment on HTTPS://ITOWNSRESEARCH.GITHUB.IO/2018_TSI_VIBES.

ROSE MATHELIER

As a developer, I worked on the conception and set-up of the global architecture of the tool. I also developed the functionalities around the edges (extraction, application of dashed and sketchy stylization), the management of the layers and the geolocation. I worked on the report, too. I became more proficient in JavaScript and three.js.

As the Scrum Master, I managed the backlog of the project and organized the tasks between the members of the group. This project helped me develop my managing skills.

LAURIE NINO

As a developer, I worked on the texturation of the faces and the objects geolocation (the way to modify x, y, z putting new coordinates). I also worked on the camera movements; modification of camera parameters (longitude, latitude, scale zoom) and creation of different points of view.

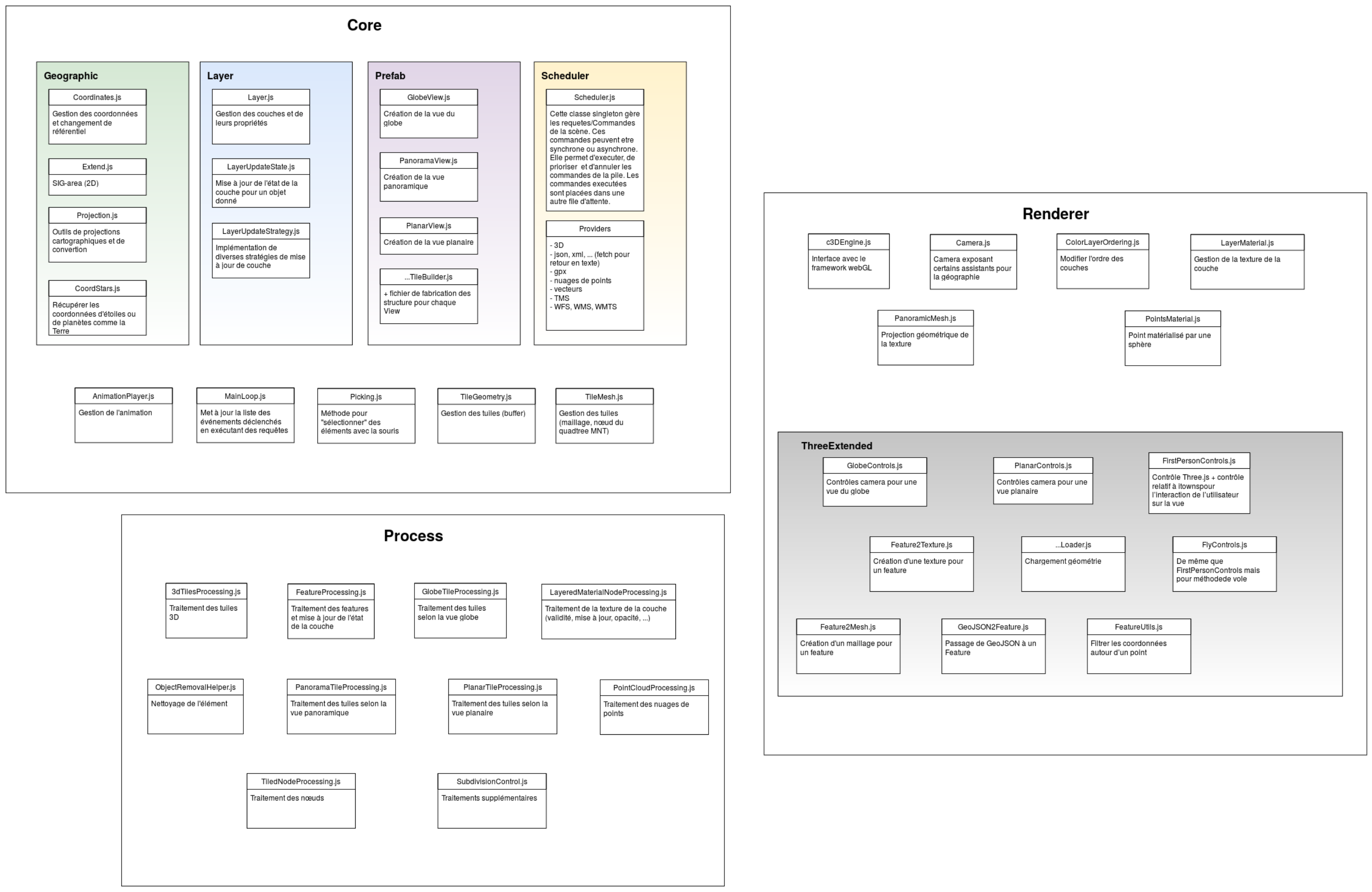
ADEL OUHABI

As a developer I took part in the writing of the unit test , and the management of the layer manager. Moreover, I integrated the loader Bati3D on our application and applied the "drag and drop" to .vibes style file to apply it on the model and finally I worked on different bugs met on the dat.GUI interactions.

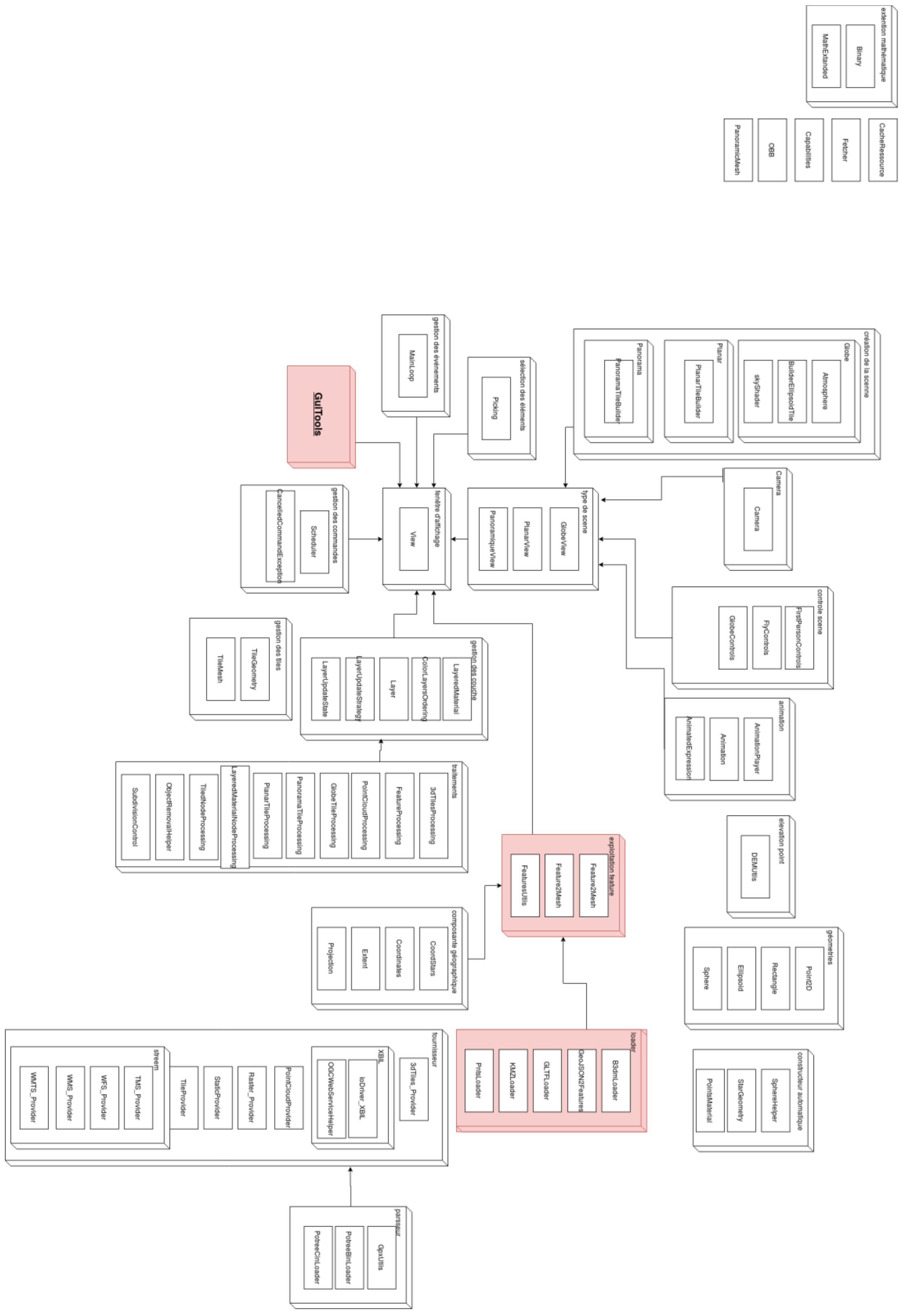
LUDIVINE SCHLEGEL

I started the project with the analyse of the current architecture of iTowns and its schema with Laurie, then I made the first version on the style file '.vibes'. Then I added a Continuous integration on TravisCI for VIBES and manage all the problems to make it work. I helped Rose for the architecture set up to split all the functions on different class. Then I studied the iTowns-legacy BATI3D loader and make our BATI3D loader and the same for the BD Topo and the iTowns example. I updated the Symbolizer and the LayerManager to adapt it to the BDTopo and BATI3D with Adel and fixed all the bugs with Rose for the BATI3D and the BDTopo. I made the objects on scene clickable to check them on the menu.

Annex 1:


Core
Geographic
Coordinates.js
Gestion des coordonnées et changement de référentiel
Extend.js
SIG-area (2D)
Projection.js
Outils de projections cartographiques et de convertion
CoordStars.js
Récupérer les coordonnées d'étoiles ou de planètes comme la Terre
Layer
Layer.js
Gestion des couches et de leurs propriétés
LayerUpdateState.js
Mise à jour de l'état de la couche pour un objet donné
LayerUpdateStrategy.js
Implémentation de diverses stratégies de mise à jour de couche
Prefab
GlobeView.js
Création de la vue du globe
PanoramaView.js
Création de la vue panoramique
PlanarView.js
Création de la vue planaire
...TileBuilder.js
+ fichier de fabrication des structure pour chaque View
Scheduler
Scheduler.js
Cette classe singleton gère les requetes/Commandes de la scène. Ces commandes peuvent etre synchrone ou asynchrone. Elle permet d'executer, de prioriser et d'annuler les commandes de la pile. Les commandes executées sont placées dans une autre file d'attente.
Providers
- 3D
- json, xml, ... (fetch pour retour en texte)
- gpx
- nuages de points
- vecteurs
- TMS
- WFS, WMS, WMTS
AnimationPlayer.js
Gestion de l'animation
MainLoop.js
Met à jour la liste des événements déclenchés en exécutant des requêtes
Picking.js
Méthode pour "sélectionner" des éléments avec la souris
TileGeometry.js
Gestion des tuiles (buffer)
TileMesh.js
Gestion des tuiles (maillage, nœud du quadtree MNT)
Process
3dTilesProcessing.js
Traitement des tuiles 3D
FeatureProcessing.js
Traitement des features et mise à jour de l'état de la couche
GlobeTileProcessing.js
Traitement des tuiles selon la vue globe
LayeredMaterialNodeProcessing.js
Traitement de la texture de la couche (validité, mise à jour, opacité, ...)
ObjectRemovalHelper.js
Nettoyage de l'élément
PanoramaTileProcessing.js
Traitement des tuiles selon la vue panoramique
PlanarTileProcessing.js
Traitement des tuiles selon la vue planaire
PointCloudProcessing.js
Traitement des nuages de points
TiledNodeProcessing.js
Traitement des nœuds
SubdivisionControl.js
Traitements supplémentaires
Renderer
c3DEngine.js
Interface avec le framework webGL
Camera.js
Camera exposant certains assistants pour la géographie
ColorLayerOrdering.js
Modifier l'ordre des couches
LayerMaterial.js
Gestion de la texture de la couche
PanoramicMesh.js
Projection géométrique de la texture
PointsMaterial.js
Point matérialisé par une sphère
ThreeExtended
GlobeControls.js
Contrôles camera pour une vue du globe
PlanarControls.js
Contrôles camera pour une vue planaire
FirstPersonControls.js
Contrôle Three.js + contrôle relatif à itownspour l'interaction de l'utilisateur sur la vue
Feature2Texture.js
Création d'une texture pour un feature
...Loader.js
Chargement géométrie
FlyControls.js
De même que FirstPersonControls mais pour méthodede vole
Feature2Mesh.js
Création d'un maillage pour un feature
GeoJSON2Feature.js
Passage de GeoJSON à un Feature
FeatureUtils.js
Filtrer les coordonnées autour d'un point

Annex 2: iTowns architecture 5/03/2018



Annex 3:

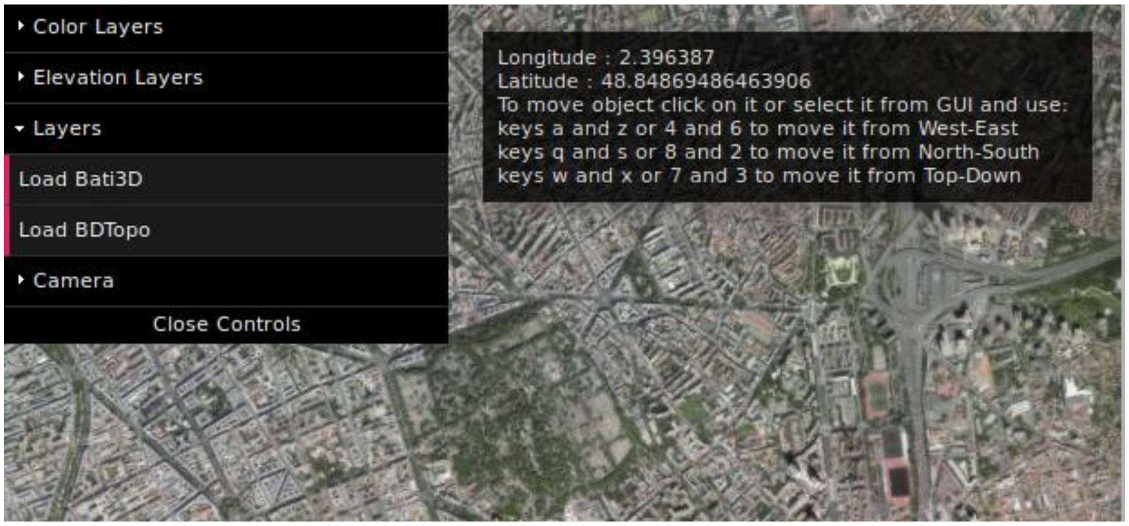
Functional Test

Todo

- things to check

INITIAL VIEW

The view look like that :



Longitude : 2.396387
Latitude : 48.84869486463906
To move object click on it or select it from GUI and use:
keys a and z or 4 and 6 to move it from West-East
keys q and s or 8 and 2 to move it from North-South
keys w and x or 7 and 3 to move it from Top-Down

TESTS FOR '.OBJ'

The test is done for one object and two objects '.obj'.

Drag&Drop '.obj' then an other.

- see the object(s) on scene
- check buton(s) added on the *Layers* menu folder
- the camera focus on the object location

Select on the menu one object and drag and drop a '.gibes' then Drag&Drop an other '.gibes', then Drag&Drop the first '.gibes'.

- 1: location object change (edges and faces)
- 2: location object change for an other place
- 3: location object change for the previous location with the same position
- each: the camera focus on the object location

Select two object and Drag&Drop a '.gibes'.

- objects on the same location

- the camera focus on the objects location

Click on the object.

- object selected on the menu if not selected before
- object not selected on the menu if selected before

Select one object and use key bord then do the same for tow object.

- use keybord and show the good move
  - keys a or 4: West
  - keys z or 6: East
  - keys q or 8: North
  - keys s or 2: South
  - keys w or 7: Top
  - keys x or 3: Down
  - other keys: nothing (arrow move the camera)

Select one object ...

- clickable button 'stylize object..', 'stylize parts...' & 'Delete layer' are add on the the *Layers* menu folder

... Select tow object ...

- only one set of button

... Unselect all.

- the clickable button are remove

Select one object ans click on 'Delete layer' then do the same with two objects.

- object(s) is(are) removed to the scene
- check buton(s) associated are removed

Select one object ans click on 'stylize object...' ...

- new folder added on the menu ('folder n')
- the folder contain:
  - clickable button: 'Save style', 'Save position', 'Load style', 'Close Symb. n'
  - checkable button: 'Display shaders'
  - sub-folder: 'Position', 'Edges', 'Faces', 'Light'

... check/uncheck 'Display shaders' ...

- show/hide shadow

... on the 'position' folder change value of each parameter ...

- rotation x: revolve around the x axis
- rotation y: revolve around the y axis

- rotation z: revolve around the z axis
- scale: change the object size
- translation x: move along the x axis
- translation y: move along the y axis
- translation z: move along the z axis
- position x, position y, position z: change the location of the object with the given coordinates (Geocentric coordinate system *WGS84 EPSG:4978*)
- reset position: place the object a its first position

... on the 'edges' folder change value of each parameter ...

- edge color: change the edges color
- edge opacity: change the edges opacity
- edge width: change the edges width
- edge style:
  - continous: continous line
  - dached: dached line
    - add parameter 'Dash Size': change the length of the dash
    - add parameter 'Gap Size': change the length between two dash line
    - the tow parameter are removed when the edge style change
  - Sketchy: complexe style line
    - add parameter 'threshold': change the limit between small and big edges style.
    - add parameter 'Stroke': change the image applaied

... on the 'faces' folder change value of each parameter ...

- opacity: change the face opacity
- color, emissive, specular, shininess: change the color and the color effect of the faces.
- texture: change the texture aplay
  - ' ': no texture
  - other : texture added
  - add parameter 'Texture': change the texture repetition

... on the 'light' folder change value of each parameter ...

- color: change the light color associated to the object
- translationx, y & z: change the light position

... click on 'Close Symb n' ...

- the symbolizer n is remove
- the object style no change
- the check button of the layer is added on the 'Layer' menu folder

... open an other 'stylize object...' ...

- the object style no change
- the parameter value coresponding at the reality

... click on 'Save style'...

- a file is saving on the computer (.vibes)

... change the style and click on 'Load style' and give the saving style or on other style for a global object then do the same with different style (edges line style, texture faces ...) ...

- the style is apply ant the symbolizer update

... click on 'Load style' and give a parts style file ...

- do nothing

... Click on 'Save position' ...

- a file is saving on the computer (.gibes)

... Do the same with two objects.

Select one object ans click on 'stylize part...' ...

- it do the same that 'stylize object...' but:
  - in the folder 'faces' it have one sub-folber by part of the object.
  - 'Display shaders' no exist

... do the same with two objects

- same nomber of part: part symbolizer open and word for the two object
- different number of part: object symbolizer open.

Select one object on the layer and drag and drop a '.vibes' ...

- the style is apply
- part style: part symbolizer open
- object style: object symbolizer open

... do the same with two objects

TESTS FOR BATI3D

Click on the 'Load Bati3D' ...

- the BATI3D is added on ths scene
- a check button is added on the 'Layer' menu folder
- the 'Load Bati3D' button is remove

... do the same that for the '.obj' with the 'Bati3D and with 'Bati3D' and '.obj' ...

differences:

- bati3D don't move (the position folder id emty for the 'bati3D' alone)
- for the 'delete layer' :
  - the bati3D is remove to the scene
  - check button is remove the 'Layer' menu folder
  - the 'Load Bati3D' button is added to the menu

TESTS FOR BDTOPO

Click on the 'Load BDTopo' ...

- the BATI3D is added on ths scene
- a check button is added on the 'Layer' menu folder
- the 'Load Bati3D' button is remove

... do the same that for the '.obj' with the 'BDTopo and with 'BDTopo' and '.obj' ...

differences:

- BDTopo don't move (the position folder id emty for the 'BDTopo' alone)
- the sketchy style line can't be apply
- for the 'delete layer' :
    - the BDTopo is remove to the scene
    - check button is remove the 'Layer' menu folder
    - the 'Load BDTopo' button is added to the menu

particularity: move the camera for ckeck the style of the other tiles no loaded.

TEST FOR CAMERA

Change all the parameter value

- reset camera: put the camera at its first place
- vue: load a define camera position and/or orientation
- longitude, latitude: change camera position ()
- zoom: change camera zoom (Geodetic coordinate system *WSG84 EPSG:4326*)