Urban data visualisation in a web browser

Jérémy Gaillard, Alexandre Vienne, Rémi Baume, Frédéric Pedrinis, Adrien Peytavie, Gilles Gesquière

Urban Data Visualisation in a web browser

Jérémy Gaillard [1,2] Alexandre Vienne [1] Rémi Baume [1] Frédéric Pedrinis [1] Adrien Peytavie [1] Gilles Gesquière [1]
[1] Université de Lyon, LIRIS, CNRS, UMR5205, France
[2] Oslandia, Tour de l'Horloge, 4 place Louis Armand, 75012 PARIS, France
gilles.gesquiere@liris.cnrs.fr

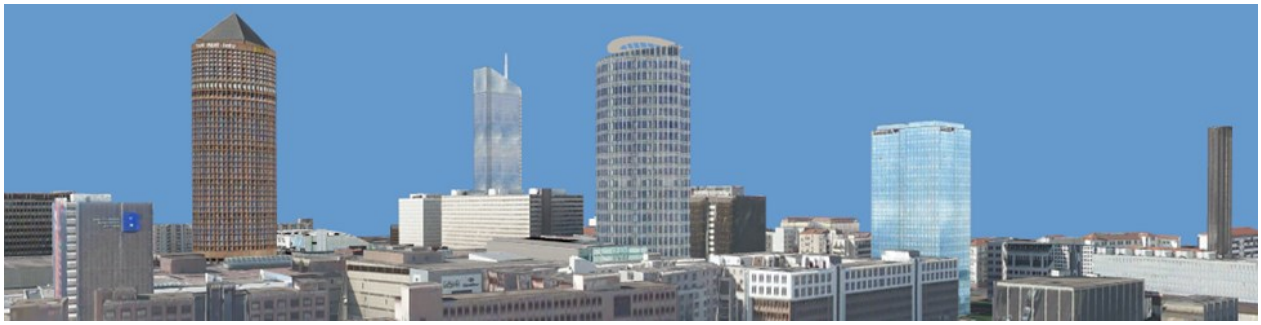

Figure 1: *Virtual model of the city of Lyon visualized in a web browser.*

Abstract

CityGML is a recent standard developed to describe, store and exchange virtual city models. Numerous software programmes have been proposed to construct, edit, modify and visualize city models, but visualisation in a web browser is still challenging. In this paper we propose a framework based on standards for visualising a large amount of 3D city data. CityGML files are processed automatically to provide a city model composed of geometries, textures and semantics. Exchanges follow the pending Open Geospatial standard named 3D portrayal. In this paper, we also demonstrate that a solution where semantics and geometries are exchanged together is possible. An effort has been made to show that an approach based on progressive textures may also be possible.

CR Categories: I.3.8 [Computer Graphics]: Applications; H.3.5 [Information Systems]: Online Information Services—Data sharing; H.2.8 [Information Systems]: Database Applications—Spatial databases and GIS;

Keywords: 3D Virtual City, WebGL, Spatial information, Standards

1 Introduction

Producing 3D geo-referenced data is now possible with accurate and semi-automatic processes based on aerial or terrestrial acquisition campaigns. Many cities own their virtual double like Lyon, Rotterdam or Berlin, for instance. A virtual model of a city requires standards to be shared and uses a significant amount of data. This represents a barrier for the spread of these 3D virtual models, since the development of such standards is still in its infancy and the storage size of this data complicates exchanges. CityGML is a standard, which has been developed by the Open Geospatial Consortium (OGC) since 2008, for exchanging virtual 3D cities. This standard seems promising but still remains mainly used by universities: there is no mainstream solution to manage and visualize CityGML files. In addition, these files are large to store because of their respect for the semantics related to 3D city objects and formatting based on the heavy XML. Full resolution textures represent a large part of this amount of data. Therefore, smooth navigation in a 3D mock-up using CityGML may not be directly achievable. It is then necessary to find a solution which would allow us to share 3D virtual models of cities while maintaining the richness, in terms of semantic information, of models contained in CityGML files. This solution should also be able to propose different modes of representation according to the needs of potential users. For instance, the view of the same city may need to be different for a tourist and for a town planner. In this 3D mock-up it may also be possible to aggregate additional data available in datastores such as 3D points from LiDAR or 2D shapes (points, lines, polylines or surfaces).

Sharing data continues to be a challenging problem. Data may be provided in open access by cities. For instance, Lyon gives access to around 550 square kilometres in CityGML, but a citizen may still have difficulty visualizing several decades of Gigabytes. It is also difficult to provide an entire view of such a dataset. Therefore, it is interesting to propose a solution based on a web browser which doesn't require the installation of any plugins to give an easier access to the data.

In this paper, we propose a framework, *Urban Data Viewer* (UDV), built heavily around standards, allowing the viewing of urban data in a web browser. Access to semantic information is possible as both the geometry and semantic information are retrieved from the server. A configuration process lets us easily set up different representation of a same city depending on the user's needs. Our solution is able to load the most relevant information first, as defined by what we call a "strategy". The use of standards guarantees the compatibility of our framework with a number of open data servers.

This article is structured as follows: we start by reviewing state of the art web city viewing techniques (Section 2), then we make a short presentation of the standards used in our framework (Section 3), leading into the description of our proposed architecture (Section 4). We propose an implementation of this architecture and evaluate our results (Section 5), before concluding our paper and discussing possible future work (Section 6).

2 State of the art

2.1 3D rendering on the web

The visualisation of urban data on the web raises a broader issue: the rendering of 3D content on the web. A number of emerging technologies have been developed in the last few years. A complete state of the art has been proposed by Evans et al. [Evans et al. 2014]. In this paper, the authors propose a classification of 3D rendering methods; like 2D web graphics, the existing 3D rendering methods can be classified into two categories: declarative methods and imperative methods (Figure 2).

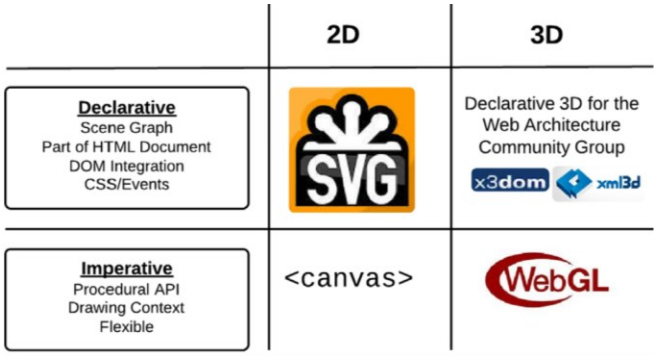

Figure 2: *Declarative vs. imperative approaches to web-based graphics (Extracted from [Evans et al. 2014]).*

Declarative methods are directly integrated into the Document Object Model (DOM), they are highly interoperable and usually have a fixed rendering pipeline. Imperative methods use a procedural API and are, in contrast, more flexible. The X3DOM [Behr et al. 2009] and XML3D [Sons et al. 2010] formats are the two popular standards for declarative 3D browser-based rendering. Both use an XML-inspired syntax. To our knowledge, the use of progressive textures in X3DOM or XML3D has not been addressed yet.

Imperative methods, which constitute the second approach, can be divided in two groups. The first requires plug-ins, such as Flash, SilverLight or Unity [Zhou et al. 2006], to work. Unfortunately, this kind of method is heavily platform-dependent, Apple's refusal to port Flash to iOS being a harsh reminder of this.
WebGL is part of the the second, cross-platform approach. It is a standard proposed by the Khronos group, an adaptation of the OpenGL ES API, which allows the programmer to access the GPU directly from the browser via JavaScript. WebGL being a low-level API, it is no surprise that libraries proposing higher-level APIs have been developed. Probably the most popular one is ThreeJS, which proposes easy to use functions for rendering 3D scenes.

2.2 Virtual 3D City

Web 3D city viewers are a trending usage of the new 3D visualisation capabilities of browsers. Several applications are already available or under development.
The well-known Google Maps has taken over the now discontinued Google Earth's 3D city viewing capabilities. It offers a very fluid experience but uses its own data and protocols, which can be detrimental to users valuing interoperability.
Cesium [Cesium] is a promising framework for the viewing of geospatial data. While it is currently open source, it also pushes its own data format CZML. Chatuverdi has proposed a solution based on Cesium to render 3D City [Chatuverdi 2014]. Cesium is attractive but is in a medium position between an open-source development and a proprietary one. Additional work may be useful to propose 3D visualisation of large datasets.
Cuardo is an application which enables the viewing of city data in 3D with a strong link to 3D databases [Cuardo]. Oslandia provides this project as open source. It relies heavily on OGC standards for its communication with data servers.
ArcGIS [ArcGIS] is ESRI's application for geospatial data viewing. It currently features scenes of 3D city models but doesn't provide an alternative to develop additional behaviours.
Another solution has been proposed by Gesquière and Manin [Gesquière and Manin 2012]. It provides a WebGL visualisation with tiled data (terrain and buildings). The authors propose a strategy to exchange only additional buildings that have not already been sent. In this solution based on JSON, it is possible to exchange geometry and the semantics linked to city objects. Unfortunately, textures are not taken into account.
OpenStreetMap offers a huge amount of crowdsourced geospatial data. Therefore, some projects such as OSM Buildings [OSMB] or ViziCities [VC] use OpenStreetMap as their database and manage to display 3D models of a great number of cities around the world. This method has the disadvantage of not allowing buildings to be textured, since OpenStreetMap does not store such information.
Fraunhofer has a web viewer that uses the declarative approach for 3D rendering: CityServer3D View Service [CSVS] describes the scene using X3DOM which is then drawn by the user's browser.
In the listed solutions, semantics and textured building are often lacking. Furthermore progressive textures are not addressed. Discussions with end users lead us to believe there is a need to prioritise the loading of the data based on their relevance. Due to these considerations, we propose in the next sections a new method.

3 Formats and standards

Coupling a 3D viewer in a client with one or several servers is a relevant challenge. An interoperable solution based on standards may be chosen as a way to take up the challenge. Solutions are currently being proposed by a joint collaboration between OGC and Web 3D consortium. An experiment was also proposed by OGC between 2010 and 2012. It demonstrated that such a solution is possible [OGC3DIE]. These initiatives test identified candidate standards like Web View Service [WVS] and Web 3D service [W3DS]. These works are currently under discussion and may lead to the proposition of a new standard, merging the two previous ones, named 3D Portrayal [3DP]. Even if this standard is not completely finalized, we will use, throughout this paper, the

proposed concepts and in particular the GetScene protocol to query the data from client to server. The delivery of this data from server to client may be done, for the geometry, with X3D, Collada or GeoJSON (Geographic JavaScript Object Notation). GeoJSON is limited to 2D, but several projects have already made their own 3D extensions. JSON can be compressed easily.
For the textures, regular formats like JPEG or PNG are used. Unfortunately, to our knowledge, no attempt has been made to provide a progressive stream for textures in a 3D city viewer in a standardized way. The Direct Draw Surface (DDS) format from Microsoft could be a good candidate. This format stores textures compressed with the S3TC algorithm. It is a widely used format and is supported by both DirectX and OpenGL. Five variations of the S3TC exist (DXT1 through DXT5). For instance, the DXT1 variation does not handle transparency but offers the best compression rate of the five. The usage of traditional image file formats on the GPU takes a lot of space on the VRAM: the GPU needs to decompress JPEG files in the VRAM in order to be able to use them. DDS files are larger than JPEG files on the disk, but can be read by the GPU while compressed, saving a lot of graphical memory.
Another interesting OGC standard to visualize geometric data on a 3D mock-up is Web Feature Service (WFS). This OGC standard offers an interface for requesting geographical features. It provides, in particular, "get" queries to retrieve features based on spatial constraints, like inside a given tile. Many processes can be carried out client-side. Providing 2D/3D vectorial data for the client to render instead of a rasterized image seems to be a good way to move geometrical analysis to the client side and to decrease server load.
Finally, CityGML is an OGC standard for modeling and exchanging urban data. It uses the XML data formalism to organize geometric and semantic data. This standard allows the exchange of data, but it is not possible to use it in a client-server application. The data is too heavy (several decades of Gigabytes) and reading XML based schemas is complex in a JavaScript application.

In this paper, we propose an approach based as much as possible on standards. The initial data is in CityGML. Each CityGML file has its geometry, texture coordinates and semantic information converted into JSON. These JSON files are provided to the client with exchange strategies inspired from 3DP. For a progressive texture mechanism, we propose a solution based on DDS. 2D vectorial data are provided with WFS thanks to the JavaScript library openlayer. This last choice brings the possibility of adding additional formats, like KML, or other datastore access in an easy and transparent manner.

4 Visualizing Urban Data in WebGL

4.1 General architecture

We present a framework based on a heavy client / light server architecture. Figure 3 shows a simplified representation of our architecture. The client and the server are developed in JavaScript. We use WebGL through the ThreeJS library to render the city.

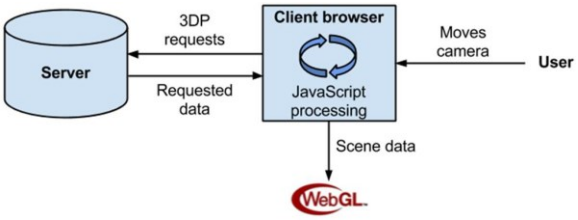

Figure 3: *General architecture of the Client-server application.*

This solution of a light server has been preferred to the one described by Gesquière and Manin [Gesquière and Manin 2012]. This new solution can manage a large number of clients.
Nowadays, the increasing capacity of client devices offers the possibility of transferring processes client side.

4.2 Preparing and providing data in the server side

In this method, we provide a solution to visualize 3D city data stored natively in a CityGML file. Data is converted and stored as files on the server in an organized manner.
Geometries are stored in JSON files, instead of CityGML, on the server. We convert all our CityGML files into this format, keeping any semantic information that could be stored in city objects. These CityGML files will have been cut into tiles beforehand with fixed size with an automatic process.
Textures also have to be converted into a specific format, DDS, to enhance the global performance of our solution.
These conversions are made with the software 3D-Use developed by our team. This pre-processing pipeline is described in Figure 4. All these pre-processes can be batched server-side. In the event of the modification of data, the tiled data may be recomputed easily and automatically.

Figure 4: *Pre-processing pipeline. CityGML files are prepared before being stored on the server.*

The server can be seen as a basic file server. It receives GetScene requests as specified in the 3DP standard and sends back the corresponding tiled data in JSON. The JSON can contain geometry, texture coordinates or semantic information depending on the requested layer. A layer is a subset of the geographical information. For example, in our case, terrain and city objects are in two different layers. We represent layers as 2D or 3D geometries bundled with semantic attributes. The server can accept separate data streams if necessary. For instance, a short term goal is to manage data provided by the 3DCityDB solution of VirtualCitySystems.
2D vectorial data can be queried via WFS on distant servers. The configuration of these streams is available client-side.
The server is structured in such a way as to be able to handle different layers and level of details. The JSON files corresponding to a layer are all grouped in the same folder that is itself stored in a folder corresponding to a specific Level of Detail (LoD). That way, layers can have different representations in each LoD.

4.3 Managing 3D City Data in a web client

The computing capacity of client devices has increased drastically in the last decade. However, it remains difficult to visualize several hundred square kilometres (around one hundred Gigabytes of data). The strategy of our method is to limit the access to a small portion of this large amount of data. Figure 5 describes our proposed architecture. It is based on two important concepts: the scheduler and the tile / texture manager.
At each frame, the client will ask the scheduler if it is ready to begin a new task. These tasks are created when the user changes the position or the orientation of the camera. They are defined by a data loading strategy, which is composed of two parts: layer management and texture management. The strategy also assigns a priority for each task. The scheduler interprets these priorities to

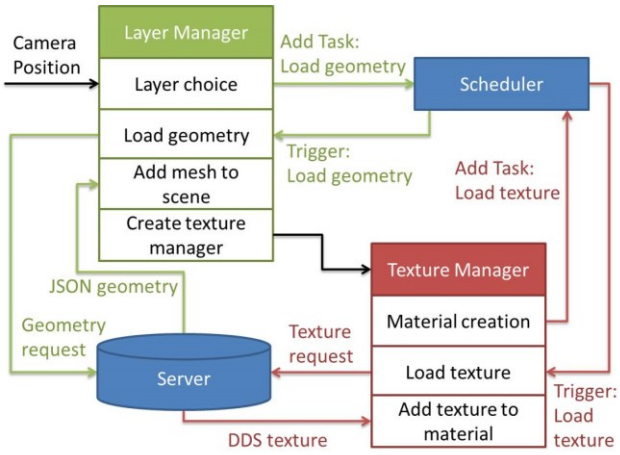

Figure 5: *Client architecture. Depending on the camera position, layers and related textures are loaded according to scheduler calls.*

decide which tasks it has to do first, while the others are stacked by order of priority and will be performed one by one.
The scheduler is built around three queues: a low priority queue, a high priority queue and a top priority queue. The top priority queue is used only for queuing unloading operations. The removal of data from memory must be done before more is added to free up space quickly and to be sure the memory won't fill up completely. The two other queues take all the other requests: geometry loading, texture loading, etc. Whether the request goes into the low-priority or high-priority queue depends on the strategy that has been defined. This strategy can be specified by the user, we will present in the fifth section the default strategy that we implemented.

The scheduler will execute every top priority request before executing high priority ones and then low priority ones. Individual queues work in a first in first out (FIFO) fashion. This strategy minimizes lags during the displacement of the user in the 3D mock-up.

According to the current position and orientation of the camera, we load a fixed number of tiles. This reduces the number of geometries simultaneously loaded in the scene. We also unload the tiles that are no longer in the current area of interest. For the scheduler, we consider that the tiles nearest to the camera should be listed as a priority.

If the virtual city model has multiple layers (DEM, Buildings, Trees, etc.), which are stored in different files, we are able to load them separately for each tile. Which layers are loaded for a tile depends on its position relative to the camera and the current strategy. The strategy can easily be adapted depending on the available layers and the user's needs. All layers' data will not necessarily be requested by the client with the same priority; displaying the DEM might be considered more urgent than displaying buildings for example. This configuration is made client-side, which allows us to provide different representations of city models with the same dataset.
Each layer can possess textures linked to its 3D geometries. Since these textures play an important role in terms of the overall performance of the viewer, we also have to choose how and when we want to load them. We have at our disposal multiple resolutions of these textures so we are able to choose different display qualities for the tiles, according to the strategy implemented. The strategy should strike a balance between performance and appearance. If the textures are not activated by the user, generic materials are applied according to the semantic information linked to 3D polygons: walls are grey, roofs red and the ground is white.
Trying to render the city only with the method presented up until now will result in severe GPU load on the client. A city model is most of the time composed of a multitude of small meshes as each building has its own. This data organization is not optimized for the GPU, as it struggles to display a vast number of unrelated meshes. To solve this issue, we merge together all the meshes from a layer. The number of meshes to manage is therefore dramatically reduced. However, this method is not without drawbacks. The semantic information of the buildings is harder to obtain since there is no longer an association between a mesh and a building. A possible way round this problem is to build an index which links each triangle to the building it belongs to, but this is put aside for future work.
We also use Mipmaps [Williams 1983] to allow progressive texture rendering. While it increases texture size by ⅓, it also reduces GPU load. With progressive textures, we can change texture quality on-the-fly, making it possible to easily adapt the rendering quality to the processing power of the device.

4.4 Providing additional data to the client

Besides the 3D rendering of the city, the framework we suggest can also display various available urban data. Thanks to Openlayers, which is used in our architecture, it is possible to access numerous sources of data. For instance, by using standard WFS streams, we can fetch additional data from distant servers and view them directly on our model. Each data type is stored inside its own layer so the user can choose which data he wants to view. These layers are processed the same way as the geometry layer. Filters provided by WFS allow to propose a tiled and tuned representation of data. Point clouds are also supported; we are thus able to visualize LiDAR data that can be superimposed over other 3D geometries in our scene. In this case, data are pre-processed from LAS format to JSON, tiled and made available on the server.

5 Implementation and results

In this section, we propose to demonstrate the capacities of our client-server application. A snapshot of our client is shown in Figure 6.

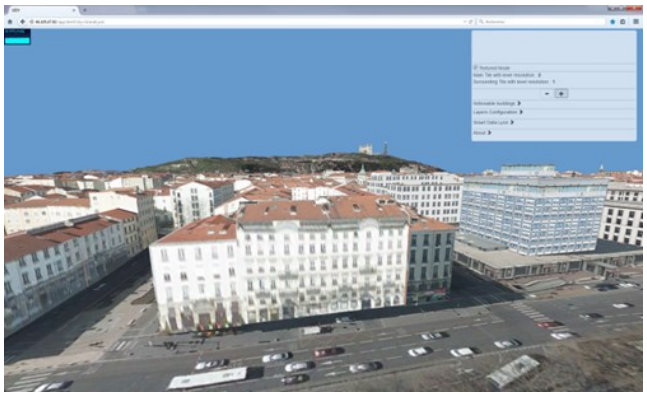

Figure 6: *A view of Lyon (France) in Urban Data Viewer.*

Tests have been made on real data provided by the city of Lyon (France) which provides 3D data covering more than 500 km² of territory (http://data.grandlyon.com/). This data is stored in

CityGML files and is divided into different layers: Buildings, Remarkable Buildings (like monuments) and Digital Elevation Models are the ones that interest us. These gigabytes of 3D textured data represent an interesting dataset on which we developed our solution. We decided to focus on the densest area; we only retained the 9 districts of the city of Lyon and the city of Villeurbanne, which represent 62 km² of data (Figure 7), to demonstrate our viewer. Our tiling process gives us the possibility of covering a larger area easily.

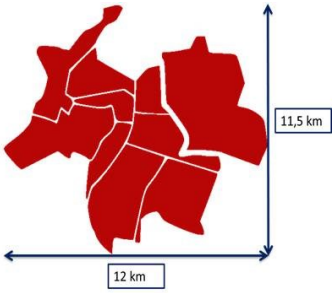

Figure 7: *The city of Lyon (France) dataset used.*

Each district is stored in a CityGML file. We set up an automatic cut according to a global and regular grid depending on the coordinate system.

A tile can contain data from two neighbouring districts, so we need to take this into account during the tiling process. With this cut, we can provide a CityGML file for each layer and for each tile. The size of these tiles can be configured according to our needs. We have established them as 500 m x 500 m squares.
We set up a rendering strategy as shown in Figure 8. The camera is on the bottom corner tile; red tiles will have all the layers while green ones will only load the DEM layer. None of the other tiles will be loaded and this tile selection process will be refreshed at each camera movement. DEM geometries are loaded as a priority without textures, and then come the geometries of the buildings. Our scheduler always begins with the tile where the camera is. After that, if textures are activated, the scheduler will load the textures of the DEM, with higher resolutions for red tiles than for green ones. It will finish by loading the textures of the buildings. If the user moves the camera before the end of all these processes, new tasks will be added at the end of the queues.

The described strategy is fully editable. The information is recorded in a configuration file stored client-side. Figure 9 is a caption of this configuration file. It contains the tiling strategy, the tiling description, and the camera location. A layer list is described afterwards. This allows some layers to be loaded in a mandatory or optional way. “Places” gives the possibility of

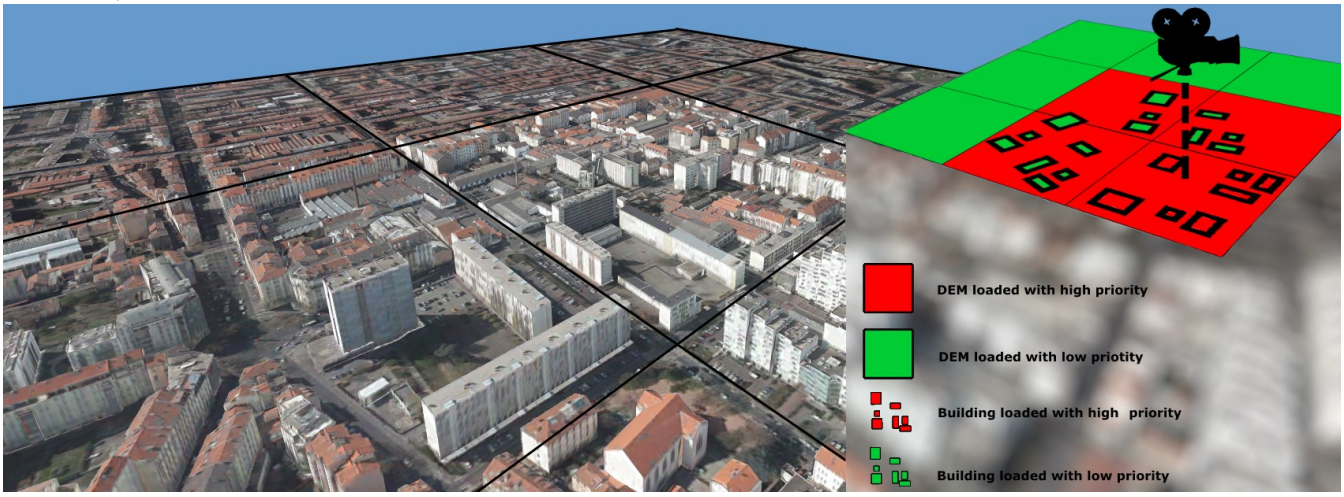

Figure 8: *Rendering strategy based on our tiled city representation.*

```
{"rangeCameraTile" : 2,
"demDistance" : 2,
"mainTuileRow" : 3688,
"mainTuileCol" : 10350,
"firstTileX" : 3675,
"firstTileY" : 10338,
"sizeTuile" : 500,
"boundingBoxSceneMin" : [1837500,5169000.000000,158.584000],
"boundingBoxSceneMax" : [1838000.000000,5169500.000000,175.933000],
"texturePath": "DDS/0/",
"tileXLimit": [3681,3698],
"tileYLimit": [10345,10359],
"cameraPosition": [6500,6000,-500],
"layersList": [["dem",true,true],["build",true,false],["remarkableBuild",true,true],["lidar",false,false]],
"cameraOrientation": [15,200,0,0],
"Places" : [{"name": "Cité internationnale", "position" : [6700,-520,8340]},
                                        {"name": "Confluence", "position" : [4419, -500, 4011]},{"name": "Part Dieu", "position" :
[6500,-500,6000]},
                                        {"name": "Notre Dame de Fourvière ", "position" : [4600,-500 ,6248]},
                                        {"name": "Lidar data", "position" : [7440, -500, 7600]}

                                       ],
"openLayerData": {"velov": ["https://download.data.grandlyon.com/wfs/grandlyon?
SERVICE=WFS&REQUEST=GetFeature&typename=pvo_patrimoine_voirie.pvostationvelov&VERSION=1.1.0&OUTPUTFORMAT=geojson&SRSNAME=EPSG:3946", false, 65535],
                                  "trees": ["https://download.data.grandlyon.com/wfs/grandlyon?
SERVICE=WFS&REQUEST=GetFeature&typename=abr_arbres_alignement.abrarbre&VERSION=1.1.0&OUTPUTFORMAT=geojson&SRSNAME=EPSG:3946",false,65280,"hauteurtotale_m"],
                                   "fontaines": ["https://download.data.grandlyon.com/wfs/grandlyon?
SERVICE=WFS&REQUEST=GetFeature&TYPENAME=epo_eau_potable.epobornefont&VERSION=1.1.0&OUTPUTFORMAT=geojson&SRSNAME=EPSG:3946",false,16777215]  }
}
```

Figure 9: *The viewer can be tuned with a configuration file stored client side.*

moving to another localisation by clicking on the given link. Finally, "OpenLayerData" describes the WFS streams that we want to access in this mock-up. Some of this information can be configured directly in the web menu provided with the client application, as we can see in Figure 10.

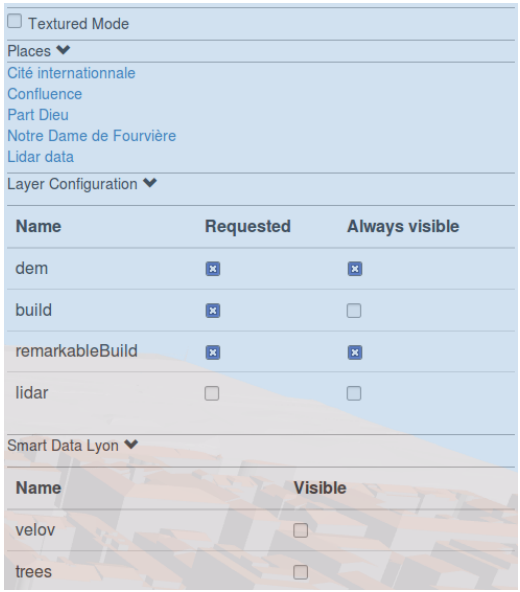

Figure 10: *Configuration menu provided with our client. The user can choose which layer he wishes to display.*

We have generated multiple texture resolutions for the data of the city of Lyon (stored in DDS, generated with the DXT1 algorithm, with Mipmap strategy). The user can choose if he wants to load textures for terrain and buildings.

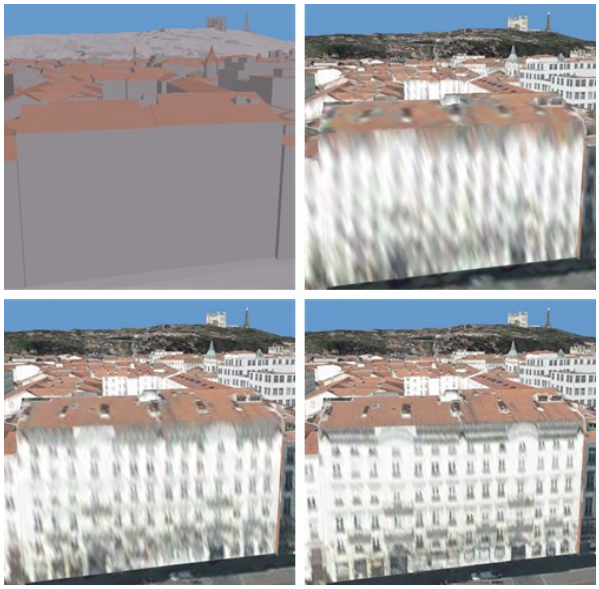

Figure 11: *Buildings and terrain can have textures of different resolutions.*

If the "Textured Mode" is activated in the menu, several resolutions are proposed. The user can switch from a low resolution to a full one. In the example of Figure 11, we zoom in on a small part of the 3D view presented in Figure 6. In the upper left-hand image, no texture is loaded. In the second image, upper right, a low resolution texture is provided. It is immediately replaced by a better one when the scheduler has time to load it (Figure 11, bottom left). In the bottom right-hand image, we have the highest texture quality. In Figure 11, we can also see that some buildings are displayed at the horizon. They are part of the Remarkable Buildings layer. Since the buildings contained in this layer are important landmarks, we experimented with a strategy that loads them from a greater distance. We did not merge remarkable buildings' meshes, so we can access its semantic data if it is available. The user can display it by clicking it (Figure 12).

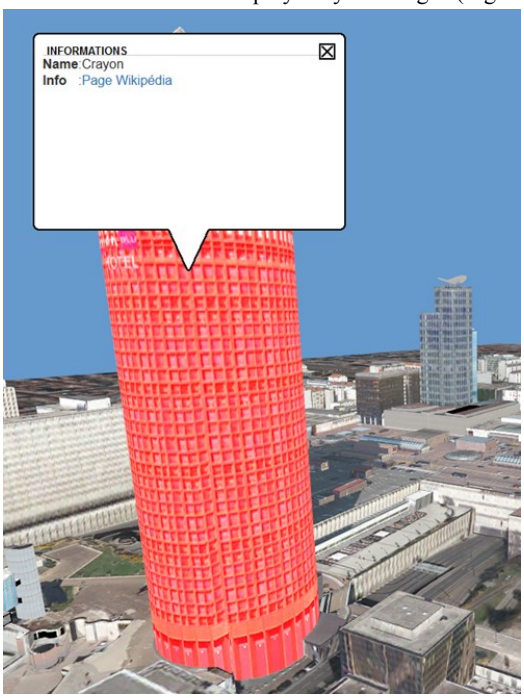

Figure 12: *Semantic data linked to a remarkable building.*

This information can be directly linked with CityGML attributes or with other data transferred into the JSON (for instance, a link to the Wikipedia page for a given building as shown here).

We are also able to view multiple layers of data. In Figure 13, we can see two layers of data directly embedded in the city model. The data was retrieved from *Smart Data Lyon*'s WFS stream. The use of a standard in the development of our framework allows us to combine data from several different sources (here our own server and *Smart Data Lyon*) with minimal effort. If the elevation linked to the 2D data is not provided, a client side process computes an elevation for each point.

Figure 13: *Symbolized trees (in green) and bike stations (blue) in Villeurbanne's "Gratte Ciel" district, France.*

As said earlier in the previous section, it is possible to make LiDAR data available in the 3D mock-up (Figure 14). In this first version, LiDAR has been tiled by a 500 m x 500 m grid.

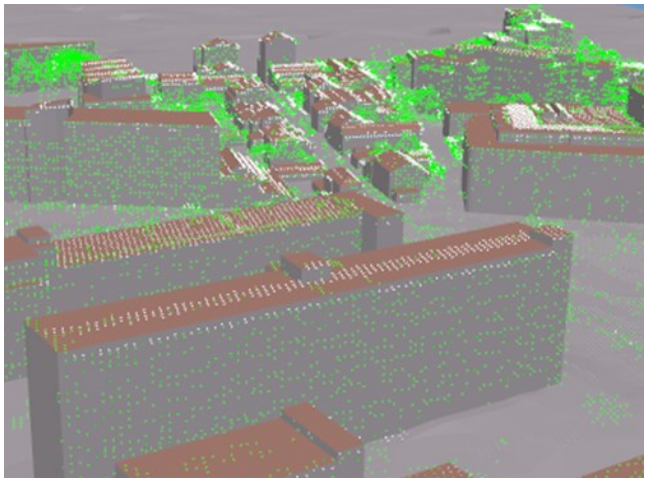

Figure 14: *Visualisation of a LiDAR layer combined with 3D buildings and terrain.*

With the data of the city of Lyon, we manage to handle more than 30 Gigabytes of data while maintaining an interactive frame rate on the client and even at a steady 60 fps when the data was fully loaded. Our scenes are composed of up to a few hundred textured buildings. Tables 1 and 2 show the performance obtained on a computer with an Intel© i5 4590 @ 3.3GHz CPU and an NVidia GTX970 GPU. Loading time is low when the data is compressed. This demonstrates that tiling based on a 500 m x 500 m grid is sufficient when we are in a dense urban district. Unfortunately, the bottleneck is for textures and LiDAR. The use of progressive textures minimizes the texture bottleneck. The Atlases proposed in the Lyon data set may also be optimized; they generally contain large bands of black pixels in order to provide square images. In the LiDAR case, the size of the tiles may be decreased to allow a more progressive load.

Our server can run on both Linux and Windows operating systems. The client is compatible with most internet browsers, including Mozilla Firefox, Google Chrome and Opera.

6 Conclusions

The advancement of web 3D standards has made it possible to display 3D content on the web without the need of plug-ins. Nevertheless, client capacity remains limited and managing hundreds of GigaBytes of geometries, semantic information and textures is still a challenging task.

In this paper, we showed that even complex scenes like cities can be rendered fluidly on the web. The framework presented in this paper allows the use of a wide variety of data available online, thanks to standards such as WFS. This is particularly useful for users who wish to seamlessly analyze data from different sources. Likewise, the use of CityGML files as input favors access to most open GIS data.

In future works, some additional optimization may be added to our framework. Most importantly, the use of Web Workers could help us make greater use of today's parallel CPU design. It could also reduce the slight stuttering the user may experience when a lot of data must be processed by keeping the application's main thread free for it to handle user commands. Streaming the geometries using progressive meshes (see [Malgo et al. 2015] for a survey of possible compression algorithm) could offer a more comfortable experience: it will enable a smoother loading of the buildings and terrain. The addition of a simple lighting system could greatly improve the overall quality of the rendering of our scenes.
As textures are such a big part of the data, we plan on doing a more thorough study of texture compression. Alternatives to DDS, such as ETC or ASTC, may be better suited for our needs and may relieve us of S3TC's patent constraints.

We are also working on modification of the geometries in order to provide multiple representations of a same city. Multiple lower Levels of Detail could be generated for buildings, in order to offer more possibilities for displaying data as proposed in [Mao 2011], [Glander and Döllner 2009]. These different representations can allow the desired building to be highlighted and reduce the global cost of scene rendering, which would also enable, for example, an increase in the number of displayed tiles.

Table 1: *Measured performance of the viewer for different data layers.*

	JSON Size	Compressed size	Geometry download	Parsing	Processing
DEM	2,2 Mb	0,341 Mb	113 ms	12 ms	57 ms
Buildings	2,5 Mb	0,401 Mb	84 ms	14 ms	32 ms
Remarkable buildings	1,2 Mb	0,187 Mb	47 ms	7 ms	15 ms
LiDAR	21 Mb	2,5 Mb	2520 ms	92 ms	17366 ms

Table 2: *Texture size and loading time.*

	Texture download							
	Resolution 0		Resolution 1		Resolution 2		Resolution 3	
	Size	Download	Size	Download	Size	Download	Size	Download
DEM	0,68 Mb	76 ms	2,05 Mb	347 ms	8,2 Mb	1175 ms	32,8 Mb	4118 ms
Buildings	1,92 Mb	3280 ms	5,75 Mb	4998 ms	23 Mb	11808 ms	106 Mb	41090 ms
Remarkable buildings	1,21 Mb	2563 ms	3,63 Mb	3624 ms	14,5 Mb	8850 ms	64,8 Mb	29160 ms

7 Acknowledgements

This work was supported by the LABEX IMU (ANR-10-LABX-0088) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR). We would like to thank the French company Oslandia who will fund the future extensions of this work through the phd thesis of Jérémy Gaillard. CityGML data are provided by "Lyon metropole" [DGL].

8 Bibliography

Behr, J., Eschler, P., Jung, Y. and Zöllner, M. 2009. X3DOM: A DOM-based HTML5/X3D Integration Model. *Proceedings of the 14th International Conference on 3D Web Technology* (2009), DOI = http://doi.acm.org/10.1145/1559764.1559784, 127-135.

Chatuverdi, K. 2014. *Web based 3D analysis and visualization using HTML5 and WebGL* (2014)

Evans, A., Romeo, M., Bahrehmand,A., Agenjo,J., and Blat, J. 2014. 3D graphics on the web: A survey. *Computers & Graphics vol. 41* (2014), DOI = http://dx.doi.org/10.1016/j.cag.2014.02.002, 43-61.

Gesquière, G. and Manin, A. 2012. 3D Visualization of Urban Data Based on CityGML with WebGL. *International Journal of 3-D Information Modeling vol. 3* (2012), DOI = http://dx.doi.org/10.4018/ij3dim.2012070101, 1-15.

Glander T. and Döllner J. 2009. Abstract representations for interactive visualization of virtual 3D city models. *Computers, Environment and Urban Systems* 33(5) (2009), 375-387.

Malgo, A., Lavoué, G., Dupont, F. and Hudelot, C. 2015. 3D Mesh Compression: Survey, Comparisons, and Emerging Trends. *ACM Computing Surveys* vol.47 (2015), DOI = http://dx.doi.org/10.1145/2693443.

Mao, B. 2011. *Visualisation and Generalisation of 3D City Models*.

Sons, K., Klein, F., Rubinstein, D., Byelozyorov, S. and Slusallek, P. 2010. XML3D: interactive 3D graphics for the web. Web3D '10: Proceedings of the 15th International Conference on Web 3D Technology (2010), DOI = http://doi.acm.org/10.1145/1836049.1836076, 175-184.

Williams, L. 1983. Pyramidal parametrics. *SIGGRAPH '83 Proceedings of the 10th annual conference on Computer graphics and interactive techniques* (1983), DOI = http://dx.doi.org/10.1145/800059.801126, 1-11.

Zhou G., Tan Z., Cen M. and Li C. 2006. Customizing Visualization in Three-Dimensional Urban GIS via Web-Based Interaction. *Journal of Urban Planning and Development* (2006), 132(2), 97-103. DOI = http://dx.doi.org/10.1061/(ASCE)0733-9488(2006)132:2(97).

[3DP] 3D Portrayal Working Group. Retrieved March 30, 2015 from http://www.opengeospatial.org/projects/groups/3dpswg

[ArcGIS] ArcGIS. Retrieved March 30, 2015 from http://www.arcgis.com/features/

[Cesium] Cesium - WebGL Virtual Globe and Map Engine. Retrieved March 30, 2015 from http://cesiumjs.org/

[CSVS] CityServer3D VIEW Service. Retrieved March 30, 2015 from http://www.cityserver3d.de/en/

[Cuardo] Cuardo. Retrieved March 30, 2015 from https://github.com/Oslandia/cuardo/

[DGL] Data Grand Lyon. Retrieved May 13, 2015 from http://data.grandlyon.com/

[OGC3DIE] 3D Portrayal Interoperability Experiment. Retrieved March 30, 2015 from http://www.opengeospatial.org/projects/initiatives/3dpie

[OSMB] OSM Buildings. Retrieved March 30, 2015 from https://github.com/kekscom/osmbuildings/

[VC] ViziCities. Retrieved March 30, 2015 from http://vizicities.com/

[W3DS] Web 3D Service. Retrieved March 30, 2015 from http://w3ds.org/

[WVS] Web View Service. Retrieved March 30, 2015 from http://www.webviewservice.org/