Applied Geomatics (2021) 13:71–87
https://doi.org/10.1007/s12518-020-00325-4

ORIGINAL PAPER

Web 3D: a CityGML viewer for cross-domain problem resolution

Marina Álvarez[1] · Javier Fco. Raposo[1] · Mónica Miranda[1] · Ana Bello[2] · Miguel Barbero[1]

Received: 29 December 2019 / Accepted: 8 June 2020 / Published online: 22 June 2020

Abstract
3D urban models are a key component of diverse applications based on geospatial data, such as urban management and urban planning. CityGML models (OGC standards) are interoperable and allow the integration and dissemination of data within the spatial data infrastructures (SDI) objectives. 3D web viewers must allow model visualization and user interaction with geospatial data. This paper presents a system architecture for web visualization of a CityGML urban model. The system allows geospatial data structuring and storage as well as model visualization and access through a Web3D viewer. During the development of the system, the cross-domain problem was solved using GeoJSON. This change in format improved user access to the 3D urban model and its data. This standard also enabled the integrated storage of geospatial information and allows access to information from other geospatial data servers without cross-domain problems.

Keywords Web3D · CityGML · GeoJSON · Geospatial Databases · Cross-Domain

Introduction

The development of spatial data infrastructures (SDIs) (Valencia and Muñoz-Nieto 2018) has supposed a revolution in managing, using and broadcasting geographical information (GI). Its standards, such as CityGML (Gröger et al. 2012), have allowed the development of interoperable and scalable 3D urban models. It is thus desirable that these models be accessible through the internet, although some issues affecting potential users can arise.

An important issue is encountered during visualization of models on the internet. The SDI visualization services (WMS or WFS) that can display 2D information work well with light and heavy clients, but the visualization of 3D urban models in CityGML is more complex. This is because the standard is designed for 3D model representation, not its direct visualization (Di Staso et al. 2015). The visualization of CityGML files is difficult due to their size, memory limitations or the lack of supporting features installed in specialized viewers (Arroyo Ohori et al. 2018). Another issue is the storage of the information, which may be in a database or in appropriate models. The former option is more flexible since it permits information to be modified without having to make changes in the urban model.

The language used in CityGML models is based on XML. These files present cross-domain problems for security reasons (Sierra 2013). This makes it difficult to download and use documents from a web source other than the one that hosts the application. To address this problem, it is necessary to use a different format that does not have such a problem.

The 3DTiles format was rejected, as it addresses storage problems but not cross-domain problems. GeoJSON (based on JSON) was the format chosen. This is a geospatial data exchange format based on JavaScript (Carter 2018), which is the language implemented by web browsers. This is why it does not present a cross-domain problem. GeoJSON also generates lighter files because it is not a markup language like XML.

The new version 3.0 of CityGML will include new encoding specifications for other formats in the next future, such as GeoJSON (Kutzner and Kolbe 2018) (Kutzner et al. 2020). The OGC is finalizing the process to include JSON among its encoding extensions (OGC JSON 2019).

In this paper, the design and implementation of a web3D application that allows access to 3D CityGML models—specifically, a 3D urban model of the historic centre of Lorca (Spain) is presented (Álvarez et al. 2018)—. At the same time, the refinement and correction of the system, in order to solve

✉ Marina Álvarez
marina.alvarez@upm.es

Mónica Miranda
monicamirandam@gmail.com

[1] Universidad Politécnica de Madrid (UPM), Madrid, Spain

[2] Universidad Complutense de Madrid (UCM), Madrid, Spain

the cross-domain problems encountered in using it, are also explained. The system was improved by implementing a CityGML-to-GeoJSON conversion tool and by modifying part of the system implementation. The aim was to find a solution that combined the ideas of other applications of the employed tools by storing CityGML data and viewing them on the internet, thereby taking a new path through the current types of existing software solutions.

The achievement of the main goal required the fulfilment of the following specific aims: (1) The concept of an urban model and information related to it is adequately presented and explained. (2) Users can access the 3D model, permitting interoperability. (3) The Web3D server is designed and implemented. (4) The 3D viewer is developed. (5) Options complementary to the viewer are included, such as the representation of data layers from different WMS servers.

This paper is divided into the following sections: the "State-of-the-art" section presents the state-of-art technology in related fields; the "Materials and methods" section describes the materials and methods used; the "Development of the system implementation" section outlines the system implementation; and in the "Results" section, the results are ultimately described. Last, a discussion and conclusion are given.

State-of-the-art

At the end of the twentieth century, different organizations greatly developed the field of 3D information. ISO 19115 (ISO 19115 2020) was conceived, and with the Web3D Consortium (Web3D Consortium 2020), founded in 1997, several standards for the publication of 3D graphics on the internet, such as X3D (Extensible3D) (X3D 2020), were developed. The X3D standard arrived after VRML (VRML 2020); it is open, extensible and multi-platform interoperable, and it is relevant to publishing geospatial information on the internet. After that time, work on visualization solutions followed that standard (Yali and Huijie 2018).

At the beginning of the twenty-first century, the 3D Information Management Working Group of the Open Geospatial Consortium (OGC) (OGC 2020) was created, and it included CAD and GIS software providers, administrative agencies and European governments. From its beginnings, this group has worked on developing the necessary standards for forming a 3D SDI (Basanow et al. 2008) and on investigating the interoperability of 3D geospatial information.

In this context, in 2008, the term 3D SDI originated (Peláez 2018). CityGML became an OGC standard, providing a model for describing 3D objects in terms of their geometry, topology, semantics and appearance. That same year, OGC began the creation of a service called W3DS (W3DS 2019) for the distribution of CityGML 3D data, which was possible due to advances in language development, geospatial databases and distribution and visualization services for 3D data. At the same time, the format of Google Earth, KML (KML 2020) became an OGC standard due to the many users and large amount of development related to it.

Spatial databases

Regarding databases, it is worth mentioning, first, that Oracle introduced the 3D Spatial Engine in version 11 g (Oracle Spatial 2020) in 2009. Since then, it has been improved and further developed so that it can store and manage three-dimensional geometric information. In that same year, the database manager PostGree, developed by the OSGeo Foundation, added the PostGIS v2.0 extension (PostGree 2020) to support 3D element storage.

3D City DB then appeared (3DCityDB 2020); it was created by the Geoinformatics Department of TU München. It consists of an open geographic database that allows the storage, representation and management of virtual 3D city models in a standard relational spatial database. It employs the CityGML standard and includes tools to convert CityGML data to KML, COLLADA (COLLADA, wiki 2020) and gTIFF (gTIFF 2020) formats for visualization in Google Earth, ArcGIS and Cesium (Yao et al. 2018).

Data formats

Concerning services for 3D model broadcasting, some formats, such as CityGML, face some problems, such as cross-domain problems. Today, more efficient data formats exist, such as JSON and GeoJSON, for addressing these issues, permitting a document to be downloaded from a web service other than that of the host of the web application used, as previously mentioned.

GeoJSON (GeoJSON 2020) is a format for geospatial data transfer based on JavaScript (JSON). Its usage has been recently extended to cartography applications in web environments, since it permits the transfer of data in a fast, light and simple way (Sierra 2013). It also has the advantages of being simpler to read and code, generating lighter files and being a web- and mobile-friendly format (VRML 2020; OGC 2020). Its most important feature is that it avoids cross-domain problems.

Viewers

Regarding web map viewers, the first developments were made in the 1990s. In 1993, the Xerox Parq Map Viewer was born, and in 1998, TerraServer (TerraServer 2020) was created based on servers of the Windows NT Server type and their related databases of the SQL Server type. The latter

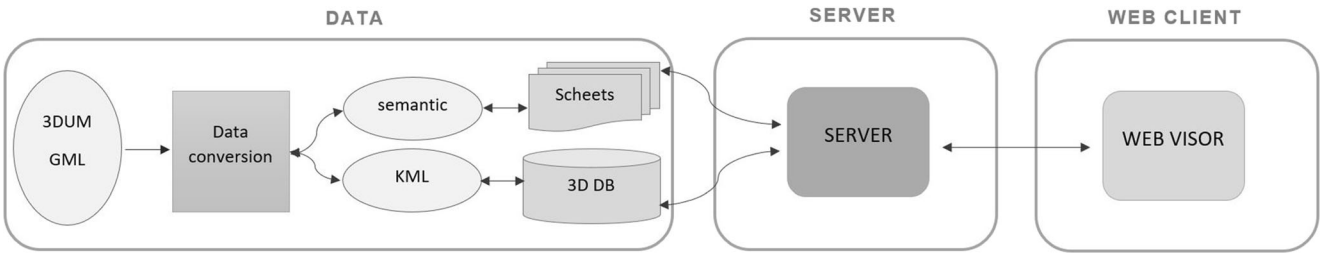


Fig. 1 Architecture of the system

model disseminated aerial images along with the first digital graphical raster, thus becoming one of the world's most popular WMS services at the time.

In the first decade of the 2000s, GeoServer was developed (GeoServer 2020), consisting of an open-code map server in the Java language, and it would achieve interoperability. It was able to read, interpret and publish data and standard services defined by OGC. In a parallel way, GeoTools (GeoTools 2020) was created; it is a free software written in Java that provides a library of utilities and implementations of the standard services proposed by OGC.

In 2005, Google created Google Maps and Google Earth as geospatial information-heavy clients. That same year OpenLayers (OpenLayers 2020) was defined, consisting of a library in Javascript dedicated to the implementation of light clients. In 2012, CartoDB (Carto 2020), later Carto, was founded. It was the first company to transfer GIS processes to the cloud using their own servers to transform data into images. Similarly, MapBox (MapBox 2020) created a new standard for sending geographical information through the net in a quicker and more efficient way.

Concerning 3D web visualization, from 2008 onwards, several viewers and free CityGML tools were developed, such as FZK Viewer (FZK Viewer, KIT 2020), LandXplorer CityGML (LandXplorer, Autodesk 2020), Aristoteles (Aristoteles, koblenz-landau University 2020), 3DGIS cityvu (Cityvu, 3DGIS 2020), libcitygml C++ library and citygml2vrml converter (citygml2vrml converter 2020) and GML Viewer (3DGIS 2020).

In recent years, an increasing amount of research on the net publication of IG 3D information has been performed (Costantino et al. 2019; Büyükdemircioğlu and Kocaman 2018; Blut et al. 2017; Prieto et al. 2016).

Materials and methods

In this work, the development of an operative visor that shows urban 3D models stored on a database, built on a web browser with free access, is described. This system follows the client/server model and permits the optimal management of all the parts employed in the CityGML urban model publication process.

A complete system has been conceived (Fig. 1), which incorporates a CityGML-to-KML conversion tool, a web server and a web client so the geometric and semantic information of the 3D model can be viewed.

Then, considering the problems identified during its first test, the system was improved (Fig. 2) so that a solution to the cross-domain problem could be installed. A tool for exporting files to GeoJSON was developed, and thus, some modifications to the system implementation needed to be made.

Materials

During the system development, the software and data described below were employed.

Input data

The input data are certain CityGML files generated in the framework of a 3D urban model of an area of Lorca's city centre based on LiDAR and cadastral data (Spanish Cadastre) (DG Catastro 2020).

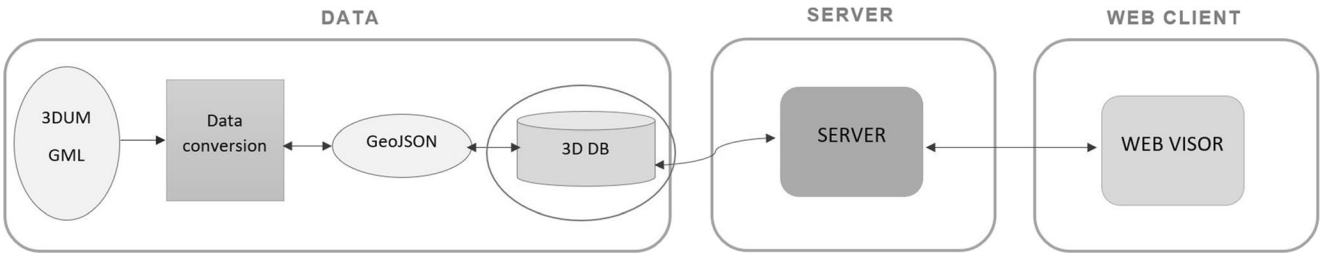


Fig. 2 Architecture of the improved system

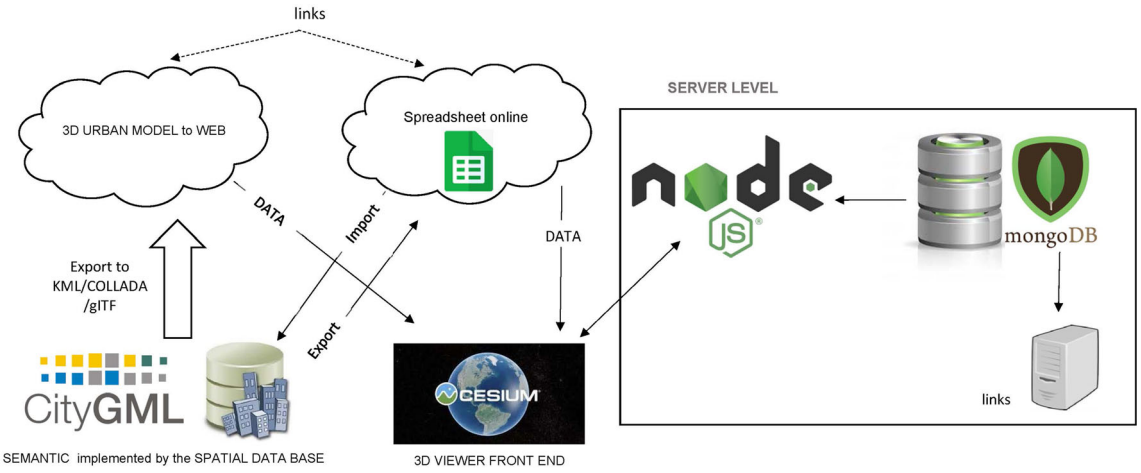


Fig. 3 System design

Software

The OGC software and open-source software solutions were used: 3DCityDB and MongoDB. 3DCityDB is an open database designed for the storage, representation and management of 3D CityGML urban models. It is based upon PostGIS, the spatial extension of PostGreSQL. It permits conversion between formats and the separation of the geometric and semantic information of the model.

MongoDB (MongoDB 2020) is a NoSQL database system that does not use tables and rows but an architecture of collections and documents that emulate their behaviour. It is the free software that performs queries in JavaScript. The queries are sent and carried out directly on the database. The user has a key to protect the database from unknown accesses. This system is easy to replicate and permits automatic configuration.

The JavaScript programming language was used, as well as the following tools (Farkas 2017): NodeJS (Node 2020) for

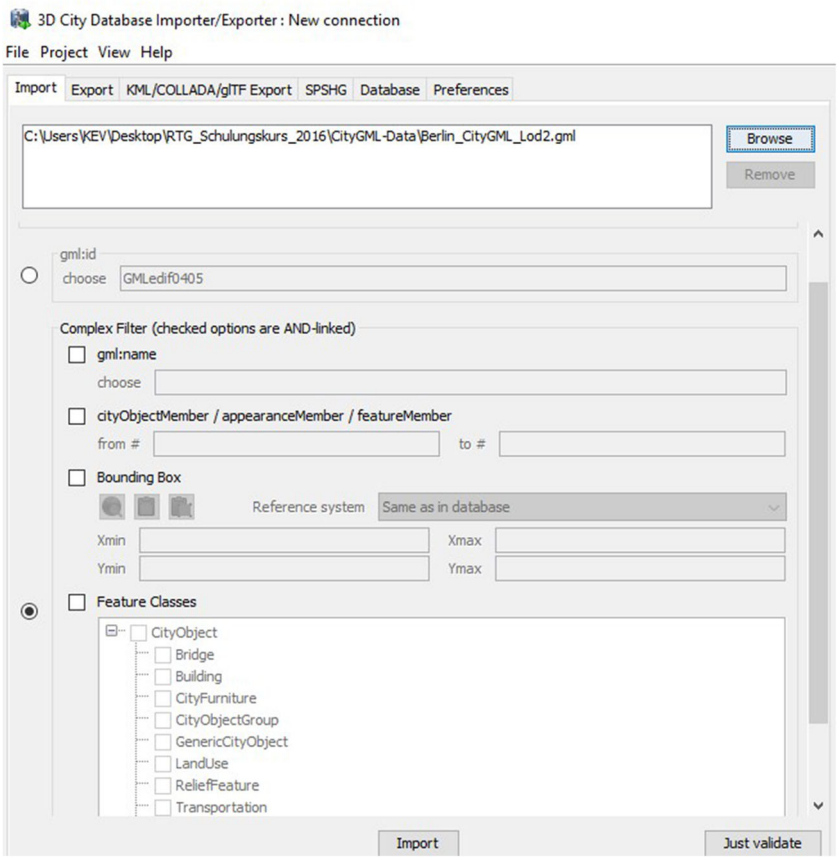


Fig. 4 Example form for CityGML-to-3DCityDB import

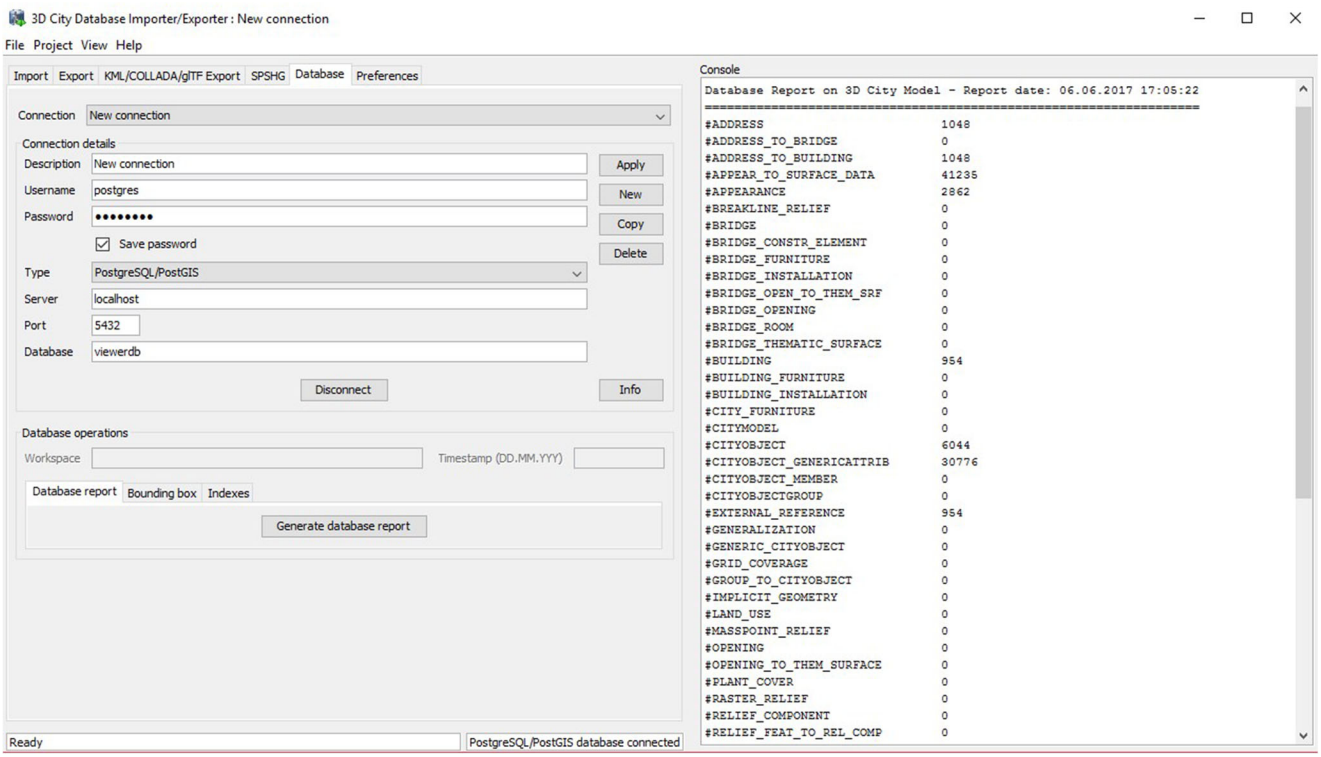


Fig. 5 Dialogue box for database checking

the service layer and JQuery (JQuery 2020) for client petitions. The JavaScript library permits simplifying interactions with the HTML documents, manipulating the DOM tree, managing events, developing animations and adding interaction with websites by means of AJAX.

Other libraries from npn used are: CesiumJS (Cesium 2020) and Spreadsheet (Spreadscheets 2020). CesiumJS (Cesium 2020) was employed for the development of the viewer. It is an open-source JavaScript library for 3D maps and globes in a web browser without add-ons. It includes other libraries: Leaflet JS (Leaflet 2020) that permits implementing and managing 2D maps and Tree JS (Tree JS 2020) employed in the 3D modelling of various objects. Spreadsheet (Spreadscheets 2020) is a free Google cloud service that permits uploading data (such as tables) to the net.

To convert KML files to GeoJSON was used Kml2Geojson (Kml2Geojson, Python 2020) was used. It is a

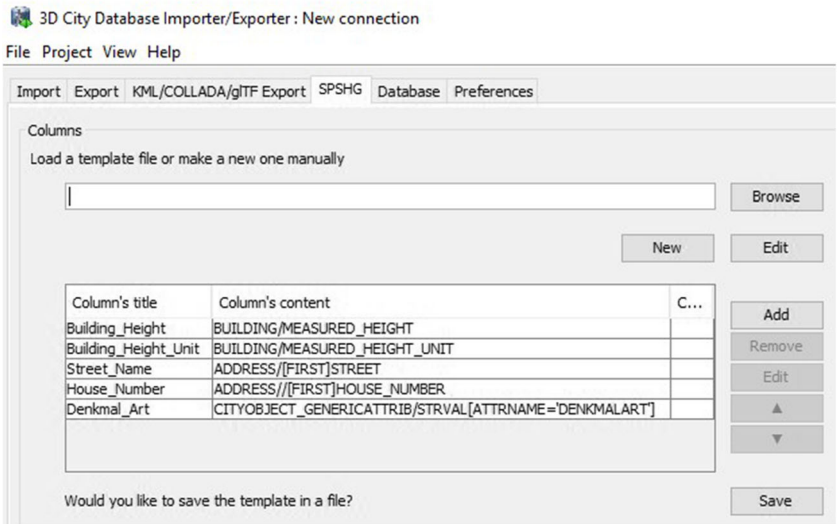
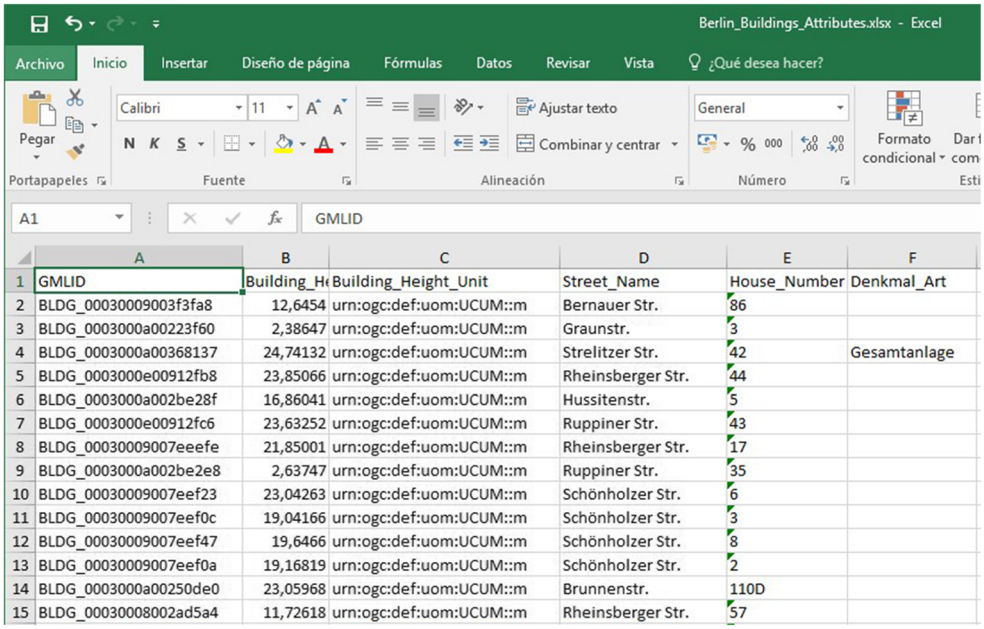


Fig. 6 Thematic data table configuration



Berlin_Buildings_Attributes.xlsx - Excel

Archivo Inicio Insertar Diseño de página Fórmulas Datos Revisar Vista ¿Qué desea hacer?

Pegar Calibri 11 Ajustar texto General Combinar y centrar Formato condicional

Portapapeles Fuente Alineación Número

A1 GMLID

	A	B	C	D	E	F
1	GMLID	Building_He	Building_Height_Unit	Street_Name	House_Number	Denkmal_Art
2	BLDG_00030009003f3fa8	12,6454	urn:ogc:def:uom:UCUM::m	Bernauer Str.	86	
3	BLDG_0003000a00223f60	2,38647	urn:ogc:def:uom:UCUM::m	Graunstr.	3	
4	BLDG_0003000a00368137	24,74132	urn:ogc:def:uom:UCUM::m	Strelitzer Str.	42	Gesamtanlage
5	BLDG_0003000e00912fb8	23,85066	urn:ogc:def:uom:UCUM::m	Rheinsberger Str.	44	
6	BLDG_0003000a002be28f	16,86041	urn:ogc:def:uom:UCUM::m	Hussitenstr.	5	
7	BLDG_0003000e00912fc6	23,63252	urn:ogc:def:uom:UCUM::m	Ruppiner Str.	43	
8	BLDG_00030009007eeefe	21,85001	urn:ogc:def:uom:UCUM::m	Rheinsberger Str.	17	
9	BLDG_0003000a002be2e8	2,63747	urn:ogc:def:uom:UCUM::m	Ruppiner Str.	35	
10	BLDG_00030009007eef23	23,04263	urn:ogc:def:uom:UCUM::m	Schönholzer Str.	6	
11	BLDG_00030009007eef0c	19,04166	urn:ogc:def:uom:UCUM::m	Schönholzer Str.	3	
12	BLDG_00030009007eef47	19,6466	urn:ogc:def:uom:UCUM::m	Schönholzer Str.	8	
13	BLDG_00030009007eef0a	19,16819	urn:ogc:def:uom:UCUM::m	Schönholzer Str.	2	
14	BLDG_0003000a00250de0	23,05968	urn:ogc:def:uom:UCUM::m	Brunnenstr.	110D	
15	BLDG_00030008002ad5a4	11,72618	urn:ogc:def:uom:UCUM::m	Rheinsberger Str.	57	

Fig. 7 Content of a thematic data table in excel format

Python (Python 2020) library that respects every KML and GeoJSON property.

Methods

First, a system design based on the particular features of the 3D CityGML models was developed, and then, it was implemented.

System design

The developed system was based on a certain kind of integrated system employed for image storage in the net, which consists of storing the net image path in the database instead of the image data itself. Thus, a three-level system was designed: Data, Server and Client.

Fig. 8 KML files from the Lorca model as seen in Google Earth

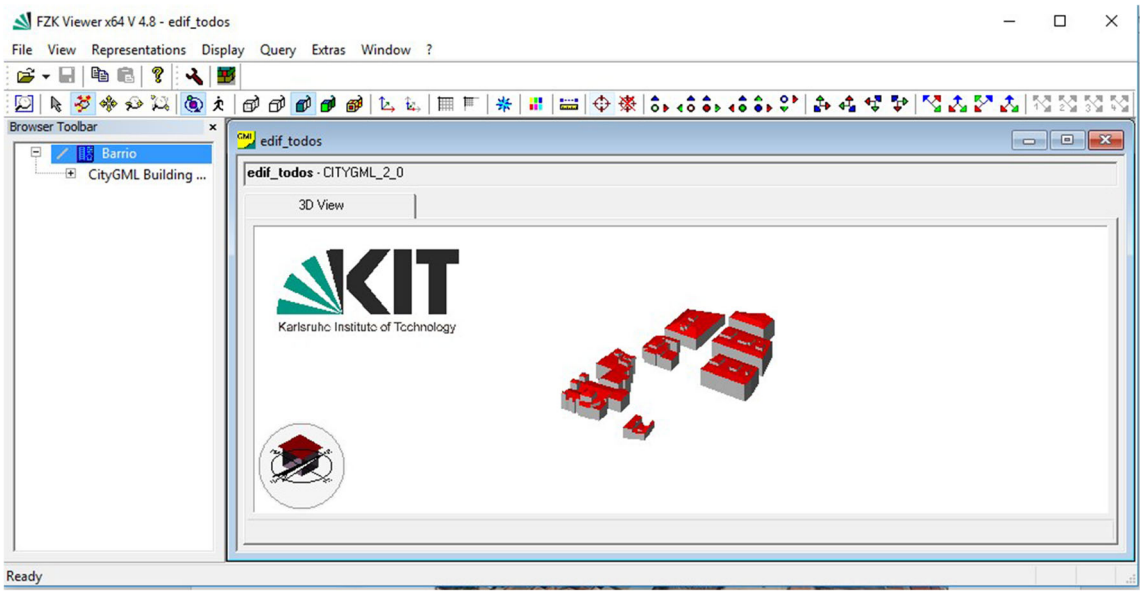


Fig. 9 CityGML files of the Lorca model as seen in FZKviewer

Data The model data are stored in an external database employed for storing, validating and transforming the CityGML data to other formats compatible with the viewer. After editing the information of the 3D model, the geometric information is exported to a NoSQL database connected at the service level. The semantic data are stored separately so they can be deployed in the viewer.

Server It ensures access to the 3D information of the models stored in the NoSQL database and their transfer to the viewer. The files should be uploaded to the net by generating logical links through which they can be accessed.

Client The graphical interface permits access to the model visualization and the semantic information as well as query realization.

The system structure is shown in Fig. 3.

The software requirement specification employed here is based on the IEEE 830 (IEEE standard 830 2020) and IEEE 29148 (IEEE standard 29148 2020) standards.

CESIUM Viewer

`new Cesium.Viewer(container, options)`

A base widget for building applications. It composites all of the standard Cesium widgets into one reusable package. The widget can always be extended by using mixins, which add functionality useful for a variety of applications.

Name	Type	Description
container	Element | String	The DOM element or ID that will contain the widget.
options	Object	optional Object with the following properties:

Name	Type	Default	Description
animation	Boolean	true	optional If set to false, the Animation widget will not be created.
baseLayerPicker	Boolean	true	optional If set to false, the BaseLayerPicker widget will not be created.
fullscreenButton	Boolean	true	optional If set to false, the FullscreenButton widget will not be created.
vrButton	Boolean	false	optional If set to true, the VRButton widget will be created.
geocoder	Boolean | Array. <GeocoderService>	true	optional If set to false, the Geocoder widget will not be created.
homeButton	Boolean	true	optional If set to false, the HomeButton widget will not be created.
infoBox	Boolean	true	optional If set to false, the InfoBox widget will not be created.
sceneModePicker	Boolean	true	optional If set to false, the SceneModePicker widget will not be created.
selectionIndicator	Boolean	true	optional If set to false, the SelectionIndicator widget will not be created.
timeline	Boolean	true	optional If set to false, the Timeline widget will not be created.
navigationHelpButton	Boolean	true	optional If set to false, the navigation help button will not be created.
navigationInstructionsInitiallyVisible	Boolean	true	optional True if the navigation instructions should initially be visible, or false if the should not be shown until the user explicitly clicks the button.
scene3DOnly	Boolean	false	optional When true, each geometry instance will only be rendered in 3D to save GPU memory.
shouldAnimate	Boolean	false	optional true if the clock should attempt to advance simulation time by default, false otherwise. This option takes precedence over setting Viewer#clockViewModel.
clockViewModel	ClockViewModel	new ClockViewModel(options.clock)	optional The clock view model to use to control current time.
selectedImageryProviderViewModel	ProviderViewModel		optional The view model for the current base imagery layer, if not supplied the first available base layer is used. This value is only valid if options.baseLayerPicker is set to true.
imageryProviderViewModels	Array. <ProviderViewModel>	createDefaultImageryProviderViewModels()	optional The array of ProviderViewModels to be selectable from the BaseLayerPicker. This value is only valid if options.baseLayerPicker is set to true.
selectedTerrainProviderViewModel	ProviderViewModel		optional The view model for the current base terrain layer, if not supplied the first available base layer is used. This value is only valid if options.baseLayerPicker is set to true.
terrainProviderViewModels	Array. <ProviderViewModel>	createDefaultTerrainProviderViewModels()	optional The array of ProviderViewModels to be selectable from the BaseLayerPicker. This value is only valid if options.baseLayerPicker is set to true.

Fig. 10 Some of the options in Cesium

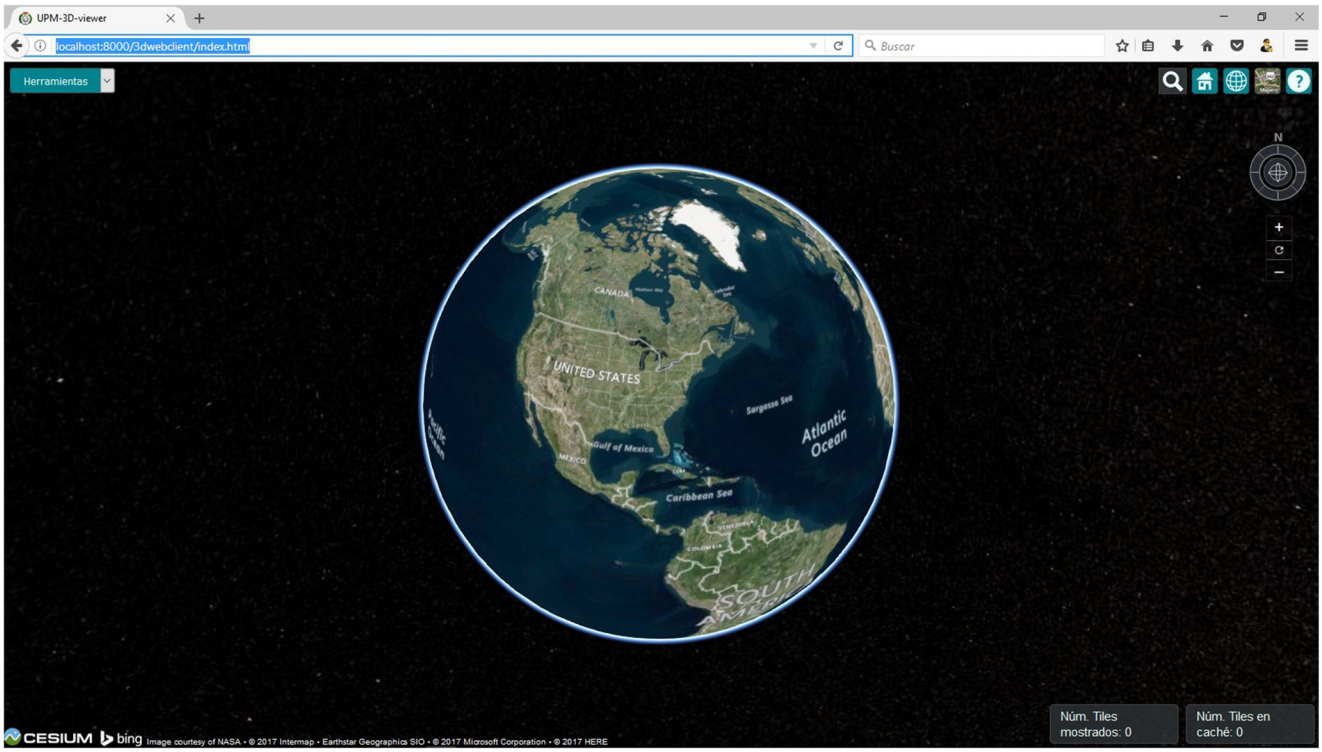


Fig. 11 Viewer home page

The system functions are spatial data management and navigation. The spatial data management includes data storage, data verification, data deletion and structured data representation. The navigation functions are graphical interfaces, navigation controls and data layer visualization management.

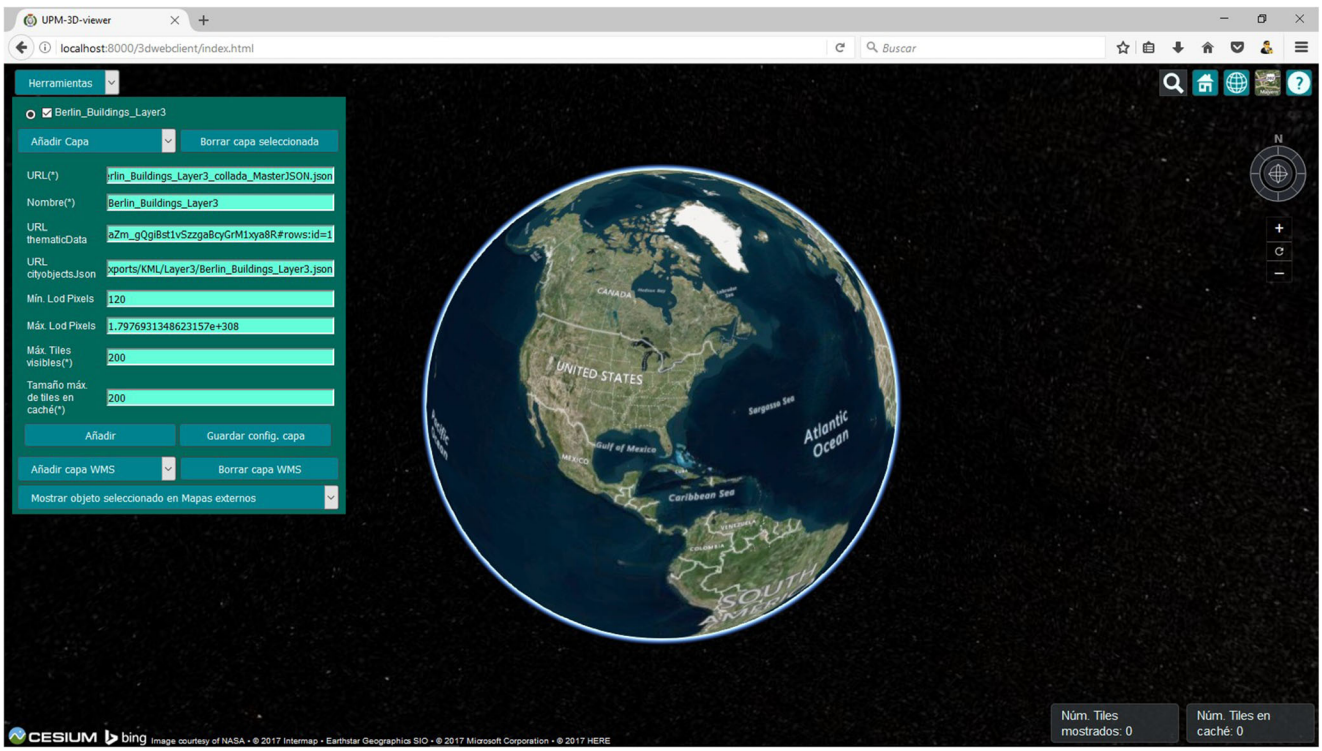


Fig. 12 Formula menu

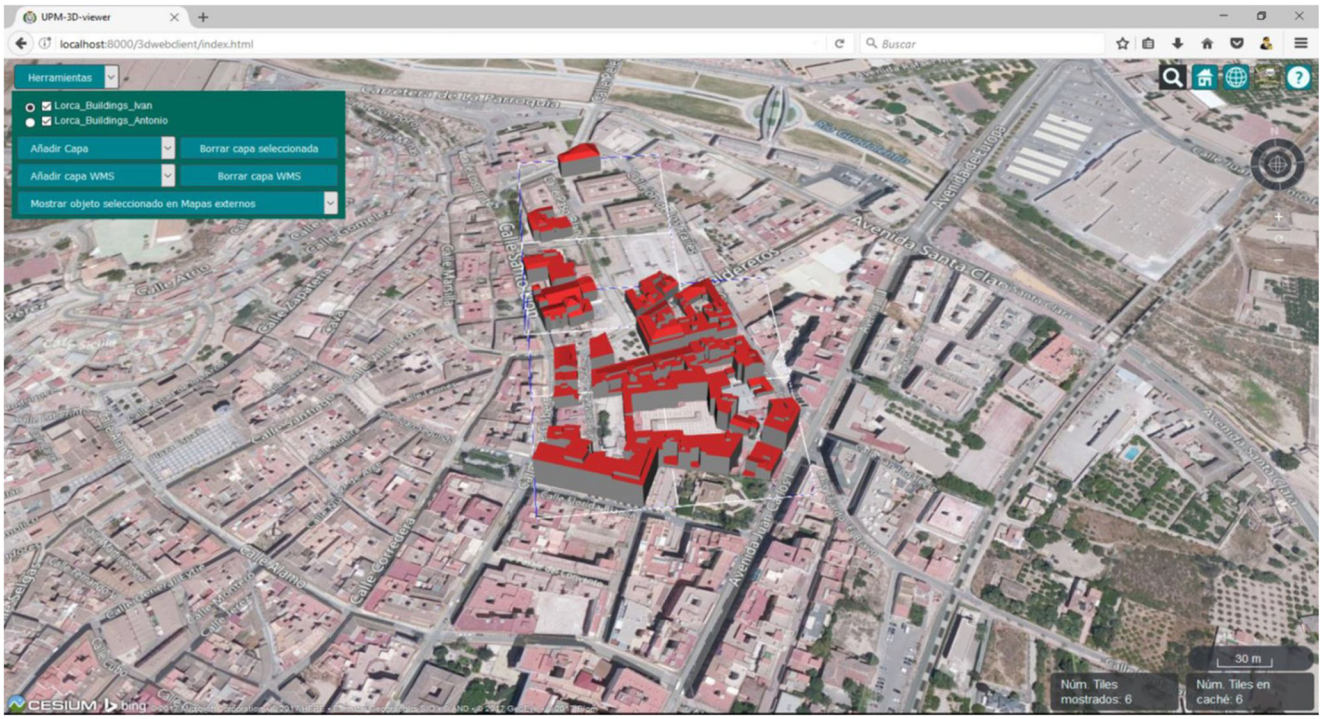


Fig. 13 Viewer with both layers active simultaneously

System implementation

The implementation of the system was performed in three phases: (1) data processing, (2) web service creation and (3) web3D creation.

Development of the system implementation

First, the starting data are loaded, which are the previously generated CityGML models of Lorca. Following this, the aforementioned steps are performed.

Data processing

Data processing was performed in four successive steps, described below.

Data import and transformation

To transform the data, the 3DCityDB database was used, which permits CityGML files to be transformed to other formats that are more adequate for web environments, such as KML, COLLADA or gITF (Kilsedar et al. 2019). Since it is based on PostGIS, PostGreSQL's geographical extension, it needs the previous installation of both tools.

The first step was to create an empty database where the imported CityGML model data were stored and their code's validity was checked (Fig. 4). The dialogue box shown in Fig. 5 helps to check the validity of the data import and the database modification process. Then, the 3DCityDB importer/exporter tool was used to transform the model's geometric information to KML.

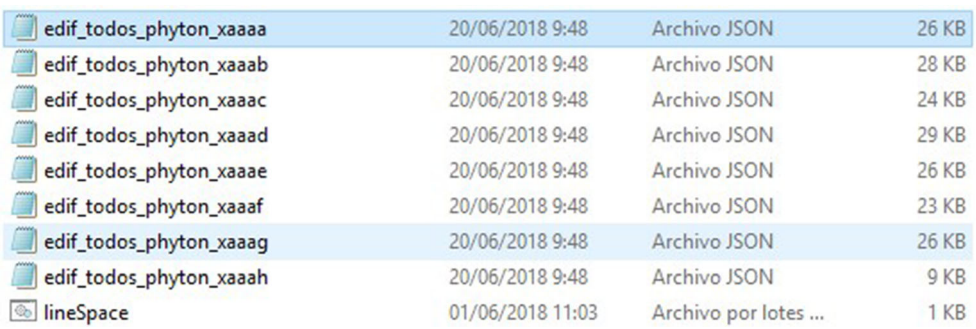


Fig. 14 Resulting data from the model data processing

Fig. 15 Part of a building

Data export

The files resulting from the transformation are exported to the MongoDB database, which is a tool connected to the server that provides 3D information.

Access to semantic information tables

3DCityDB allows the semantic information table of the model data to be accessed (Fig. 6). The information table is exported to a Google spreadsheet so that the information can be directed

Fig. 16 Complementary part of the building

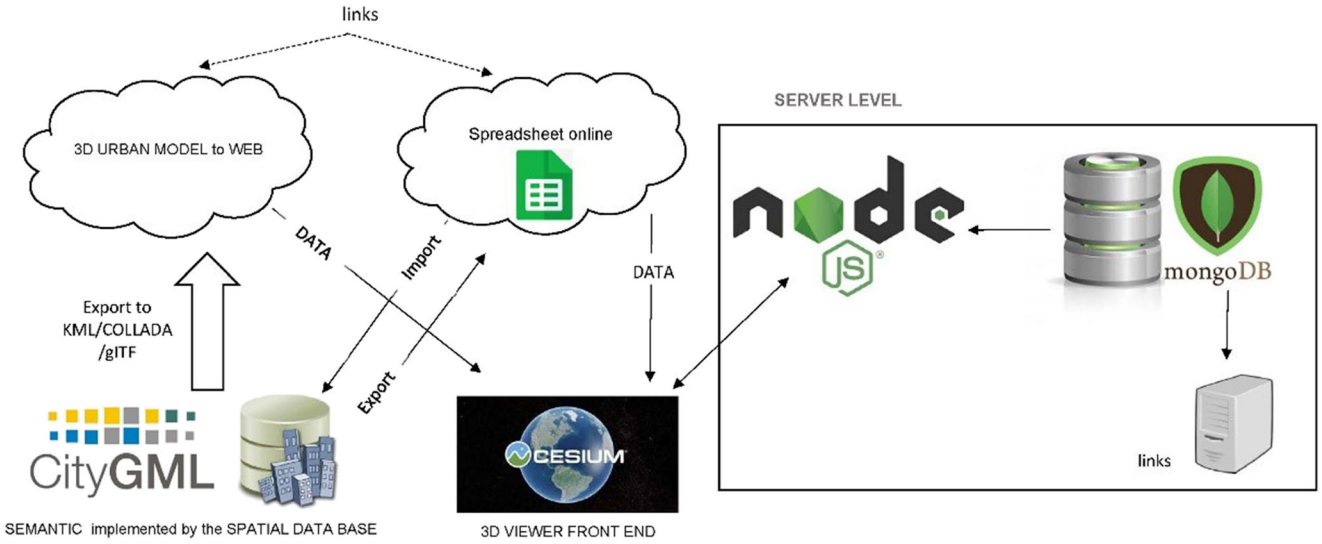


Fig. 17 Improved system design

to Web3D from the cloud. The export is bidirectional, since it is possible to create new SQL tables from the spreadsheet. Therefore, the input data are the logical links to the model and their associated thematic data table.

The table configuration can be saved as a .txt file for future use. The result is an excel file, as shown in Fig. 7. It is worth mentioning that the results vary depending on the choice of data and the kind of thematic data table.

After this process is complete, the KML files are obtained. Immediately, a double-check of the data transformation process is performed. First, the CityGML data are viewed in Google Earth (Fig. 8), and then, FZKviewer is used to perform the second check (Fig. 9). It is verified that neither errors nor information loss have occurred.

Importing data to MongoDB

The last step of data processing is importing the managed data to the MongoDB database, a non-relational database system that permits the fast search of a huge dataset. Therefore, an API Key associated with the user is established so that access to the server data query can be controlled. The system also permits the storage of information related to the IDs associated

Fig. 18 Home page of Web3D

Fig. 19 Files of the Lorca urban model transformed to GeoJSON in the developed Web3D

with the API Key account in addition to information about the location and accessing browser of the last connected device.

Web service creation

The implemented service has to be able to store the locations of the models in its database depending on the employed library. Node.js was chosen for building servers in the design of this system.

After the installation of Node.js, the following tools were used: (1) Express JS (Express JS 2020), (2) Mongoose (Mongoose 2020), because MongoDB is used, (3) The BodyParser libraries (BodyParser, npm 2020), which permit middleware to be built for configuring the format of message bodies, (4) BlueBird (BlueBird 2020), so programming can be simplified and, (5) W3C and the CORS specification (CORS, W3C 2020), which permit inter-domain communication from a web browser.

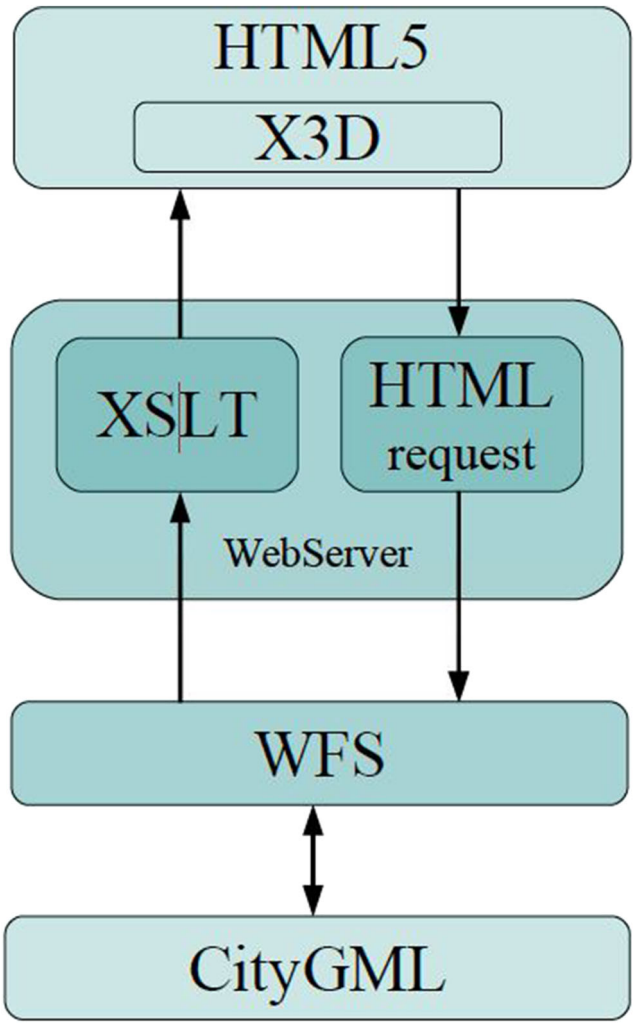


Fig. 20 Web3D GIS framework (Rodrigues et al. 2013)

Web3D creation

CesiumJS was used to develop the presentation layer of the viewer. It is a library containing numerous widgets that can be used in designing the appearance of the site where the viewer is implemented. It also permits behaviours to be configured by means of the prototype attributes (Fig. 10).

From the client, the database can be accessed with MongoDB Compass. To connect the two, some necessary JavaScript methods were developed to fulfil the base connecting functions with HTML by means of the user-associated API Key.

Results

The home page of Web3D is shown in Fig. 11. It includes a menu where the model to be loaded in the application can be selected. The user can visualize the model and perform other operations using the formula panel for queries and data insertion from the possible 3D models.

Access to the 3D models from the presentation page is obtained by using the query menu, which adds them as layers (Fig. 12). The access routes to the KML files and the corresponding thematic table need to be provided. Additionally, the viewer implements calls to WMS services.

Figure 13 shows two layers of the model simultaneously.

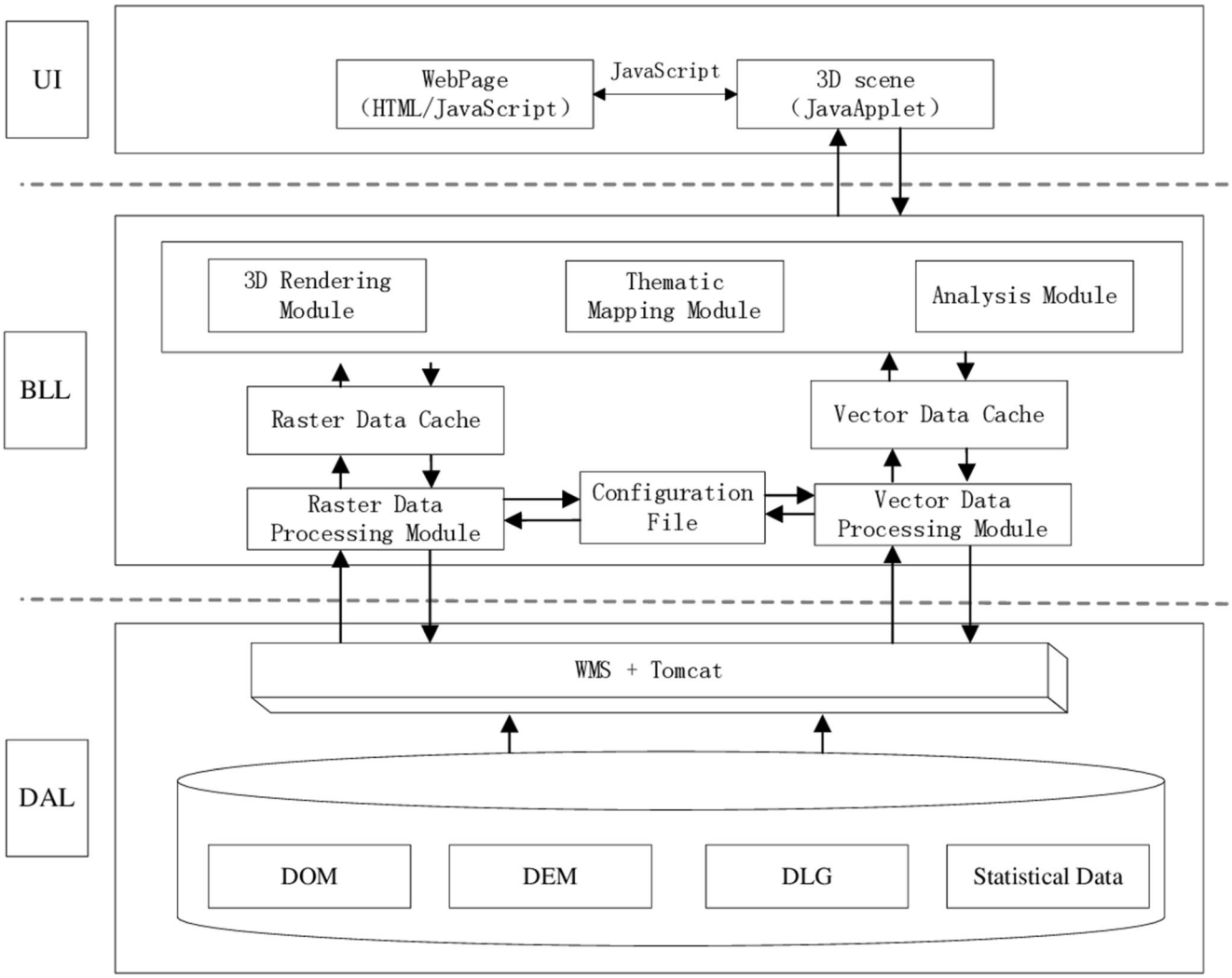


Fig. 21 Architecture of Web3D GIS (Zhu and Qiao 2013)

Although the visualization of the models was correct, some problems were detected. Although the KML format is adequate for publishing 3D models in the net because it is native to Google Earth, the results are very heavy for data exchange. Also, the transformation to KML forces the division of geometric and semantic information. As previously mentioned, the sole usage of the visor presents cross-domain problems. In this context, it acts as a protection mechanism that prevents the downloading of data from a server other than the one hosting the web application used (Sierra 2013).

Considering these problems, it was decided that the files containing 3D models would be exported to another geographic data format. GeoJSON, which is derived from the concepts of other existing spatial standards, was chosen. It simplifies coding in adapting to web environments.

GeoJSON has the following advantages: (1) it generates files of smaller size and less complexity, (2) it is a format for spatial data that reunites spatial and semantic information, (3) it is a more adequate format for the features of web environments, and (4) it is designed in JavaScript. This language is used to implement most of the existing servers and web viewers, so it avoids cross-domain problems.

To solve the aforementioned problems in the integrated system, some modifications were made to its implementation. Specifically, the data processing, web server deployment and Web3D stages were improved.

Data processing

A three-stage process was performed: KML to GeoJSON transformation, file subdivision or splitting and extension modification.

KML to GeoJSON transformation

All the KML files stored in the 3DCityDB were transformed to GeoJSON format by means of Kml2Geojson, a Python library that respects the properties of both formats. The

process is automated by applying a batch script that permits the processing of all files with a .kml extension so that they can be transformed to GeoJSON. Only a single script execution is needed.

File splitting

Before importing the files, now in GeoJSON format, they needed to be subdivided into files with a maximum size of 25 KB so the characteristics and restrictions of MongoDB could be met. This process was performed with the Geojsplit tool (Geojsplit 2020), and another script was developed for automating this process.

Extension changes

Last, given that the database accepts files with a .json extension, the names of the split files were modified to end with that extension. The .geojson-to-.json extension replacement was automated in the same manner as the two previous steps, with a script that creates JSON files. The resulting files contain the CityGML model data transformed into JSON format (Fig. 14).

Figures 15 and 16 show how some buildings can be seen only partially, with an incomplete volume, due to the file subdivision. This problem is solved when all the files are stored in the database, and thus, the complete model can be represented.

Next, the divided archives are imported to MongoDB along with a test to verify their correct visualization in the viewer.

Web service improvements

The MongoDB database was stored in the cloud net by Amazon Web Services (AWS) (Amazon Web Service 2020). AWS receives client petitions; therefore, there is no need for the system to store the files locally. The database stores data in GeoJSON format and permits files to be modified and new data to be imported in addition to receiving server petitions (Fig. 17).

Web3D improvements

To work with the database in Web3D and easily access the data starting from the home page, the following improvements were added: (1) the access function login() connects to MongoDB with the help of a user's API key. (2) The file importing function InsertCollection() permits loading a file from the server's "data_geo" directory and importing it to the database. (3) The data-loading function getcollection() loads the first found element of the database into Web3D.

Additionally, some function access buttons were added in the main window of the viewer for managing the connection to the database. *Import* button permits loading a file stored in the server. *InsertBD* function grants access to a selected GeoJSON file so it can be imported into the database. *LoadBD* function relates to the aforementioned getcollection() function.

Once these modifications are introduced, the home page of Web3D is as displayed in Fig. 18. As can be observed, the viewer has the navigation functionalities of the initial version in addition to the new data accessing functions.

Figure 19 shows the interface of Web3 loaded with Lorca's 3D City Model superimposed on an aerial photograph taken by PNOA (PNOA, IGN 2019), accessed through a connection with the implemented WMS service.

Discussion

The designed Web3D system follows a client/server architecture and consists of three levels. The division into three levels is a common structure in the design of 3D web viewers and web-GIS based on geospatial data.

At the first level, the data level, the CityGML Model is stored in a database that allows spatial information to be stored. We used MongoDB, a non-relational database. Non-relational databases can better handle the large volumes of data in Urban 3D Models. (Laksono 2018). The use of these NoSQL databases is increasing compared with the usual PostGIS or 3DCityDB (Laksono 2018) used in many developed geospatial applications (Aleksandrov et al. 2019; Prandi et al. 2015).

In the first design of our system, CityGML files are transformed into KML format, so the geometric and spatial information have to be stored separately.

The splitting of the information at this level has been seen in other Web3D studied (Tang et al. 2019, Prandi et al. 2015, Zhu and Qiao 2013, Rodrigues et al. 2013) (Fig. 20). The Web3D GIS system in the research consulted is primarily used for simulation models, thematic mapping, monitoring and basic spatial analysis. Many of the systems studied (Zhu and Qiao 2013) read CityGML data through a WFS and convert it to X3D for viewing on the internet, and they also provide tools to display and analyse geospatial information in a 3D environment (Fig. 21).

In the final design of our Web3D viewer, the CityGML files are transformed into GeoJSON format. GeoJSON supports the integration of geometric and semantic information. The data can be integrated into the JSON-based database (MongoDB) (Baralis et al. 2017). Other common databases, such as 3DCityDB, export geometric information to KLM/COLLADA (Prandi et al. 2015), and the semantic information has to be stored in other formats (Zhu and Qiao 2013).

The objectives of the implemented system are to visualize CityGML models optimally and to be able to access information from other agency servers related to city management and

maintenance without cross-domain problems. To achieve these objectives, a single server was developed (Fig. 2) that allows all the files to be published—raster, vector and DEM—in an integrated way. The publication of the models would not be optimal using the architecture proposed by Web3D GIS through a WMS for raster data and a server developed with Tomcat (e.g.) or a WFS for vector data (Rodrigues et al. 2013) (Fig. 21).

The user interaction with the model is improved by solving the cross-domain problems of other formats, such as CityGML and KML (Prandi et al. 2015). The application is innovative because it facilitates interoperability with the CityGML model by eliminating the problem of the domain (Sun et al. 2019), which allows access to information from other geospatial data servers.

CesiumJS was used for the Web3D viewer. Today, it is the best option (3D globe map) for 3D geospatial information visualization (Aleksandrov et al. 2019, Prandi et al. 2015). As GeoJSON is compatible with JavaScript (Ledoux et al. 2019), which is the language used by Cesium, the use of this format improves the visualization of the model.

Conclusions

This paper shows the development of an integrated system for publishing 3D urban models of the town of Lorca, allowing them to be visualized via a web browser through accessible data stored in a functioning database. This development, although it does not present any difficulty on a conceptual level, is somewhat complicated to implement in practical terms during some of the necessary processes described, since the system required some modifications during its implementation.

The disadvantages faced in publishing CityGML models on the net required transforming the data used to KML and then to GeoJSON as the final format. The latter is more adequate for web environments and permits faster model data transfer than the CityGML format. In addition, the transformation to JSON allows model data to be downloaded without presenting cross-domain problems.

To store the exported files in KML and GeoJSON, a non-relational database, in this case MongoDB, was used. It admits files from different formats, and it is adequate for very large quantities of data, which is common in 3D City Models.

A web application able to store and visualize data models in GeoJSON was deployed on an Amazon Web Services database. Thus, scalability was taken into account through the infrastructure of cloud services. This allows access to the data from different clients in a safe, secure and concurrent manner, and it allows a register of all queries made in the database to be created. Issues such as security and data access are thereby addressed. This development of diverse implementations did not incur problems and was performed simply, so the adequacy of the chosen software is evident.

The designed Web3D viewer improves the interoperability of the CityGML model by solving cross-domain problems. The developed system allows access to information from other geospatial data servers. In addition, the GeoJSON format allows geospatial information to be stored in an integrated way, which is suitable for geospatial information-based web applications.

In future work, an improvement of the design of Web3D is proposed that augments the precise interface to allow the addition and modification of information, taking into account the newly implemented functions for data management.

As a final conclusion, given the described circumstances, the development of tools allowing access to 3D models is necessary even today. These tools have to enable advanced analytical and processing capabilities in a light, intuitive and easily interactable navigation environment. The proposed work aims to address this need.

Acknowledgements The authors of this paper would like to thank the students from ETSI Informáticos (UPM) Kevin Cubero and David Flores for their collaboration.

Funding information This research did not receive any specific grant from funding agencies in the public, commercial or not-for-profit sectors.

References

Aleksandrov M, Diakite A, Yan J, Li W, Zlatanova S (2019) Systems architecture for management of BIM, 3d GIS and sensors data. ISPRS Ann Photogramm Remote Sens Spatial Inf Sci IV-4/W9:3–10. https://doi.org/10.5194/isprs-annals-IV-4-W9-3-2019

Álvarez M, Raposo JF, Miranda M, Bello AB (2018) Metodología de Generación de Modelos Virtuales Urbanos 3D para ciudades inteligentes. Informes de la Construccion 70(549):3–13. https://doi.org/10.3989/id.56528

Arroyo Ohori K, Biljecki F, Kumar K, Ledoux H, Stoter J (2018) Modeling cities and landscapes in 3D with CityGML. In: Borrmann A, König M, Koch C, Beetz J (eds) Building information modeling. Springer, Cham. https://doi.org/10.1007/978-3-319-92862-3_11

Baralis E, Dalla Valle A, Garza P, Rossi C, Scullino F (2017) SQL versus NoSQL databases for geospatial applications:3388–3397. https://doi.org/10.1109/BigData.2017.8258324

Basanow J, Neis P, Neubauer S, Schilling A, Zipf A (2008) Towards 3D spatial data infrastructures (3D-SDI) based on open standards—experiences, results and future issues. In: Van Oosterom P, Zlatanova S, Penninga F, Fendel EM (eds) Advances in 3D Geoinformation systems. Springer, Berlin, pp 65–86

Blut C, Blut T, Blankenbach J (2017) CityGML goes mobile: application of large 3D CityGML models on smartphones. Int J Digital Earth 12:25–42. https://doi.org/10.1080/17538947.2017.1404150

Büyükdemircioğlu M, Kocaman S (2018) A 3D campus application based on city models and WebGL. ISPRS - international archives of the photogrammetry, remote sensing and spatial information sciences. XLII-5. https://doi.org/10.5194/isprs-archives-XLII-5-161-2018

Carter PA (2018) SQL server advanced data types: JSON, XML, and beyond. Apress, Berkeley. https://doi.org/10.1007/978-1-4842-3901-8

Costantino D, Angelini MG, Alfio VS, Claveri M, Settembrini F (2019) Implementation of a system WebGIS open-source for the protection and sustainable management of rural heritage. Appl Geomat 12:41–54. https://doi.org/10.1007/s12518-019-00275-6

Di Staso U, Prandi F, Soave M, Devigili F, Amicis R (2015) 3D web visualization of huge CityGML models. ISPRS Int Arch Photogramm Remote Sens Spatial Inf Sci XL-3/W3. https://doi.org/10.5194/isprsarchives-XL-3-W3-601-2015

Farkas G (2017) Applicability of open-source web mapping libraries for building massive web GIS clients. J Geogr Syst 19:273–295. https://doi.org/10.1007/s10109-017-0248-z

Gröger G, Kolbe T, Nagel C, HÄfele K (2012) OGC City Geography Markup Language (CityGML) Encoding Standard. Open Geospatial Consortium Inc. (OGC), Wayland

Kilsedar CE, Fissore F, Pirotti F, Brovelli MA (2019) Extraction and visualization of 3D building models in urban areas for flood simulation. In: Arch. Photogramm. Remote Sens. Spatial Inf. Sci., vol XLII-2/W11, pp 669–673. https://doi.org/10.5194/isprs-archives-XLII-2-W11-669-2019

Kutzner T, Kolbe T (2018) Citygml 3.0: Sneak preview. In: Kersten TP, Gülch E, Schiewe J, Kolbe T, Stilla U (eds) Publikationen der Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation (DGPF) e.V, Munich, pp 835–839

Kutzner T, Chaturvedi K, Kolbe TH (2020) CityGML 3.0: new functions open up new applications. In: Journal of Photogrammetry, Remote Sensing and Geoinformation Science, vol 88, pp 43–61. https://doi.org/10.1007/s41064-020-00095-z

Laksono D (2018) Testing Spatial Data Deliverance in SQL and NoSQL Database Using NodeJS Fullstack Web App 1, 1–5. https://doi.org/10.1109/ICSTC.2018.8528705

Ledoux H, Arroyo OK, Kumar K, Dukai B, Labetski A, Vitalis S (2019) CityJSON: a compact and easy-to-use encoding of the CityGML data model. Open Geospatial Data Softw Standards 4:1–12. https://doi.org/10.1186/s40965-019-0064-0

Peláez A (2018) Evolución de los mapas en la Web. Mapping, vol 27, 192, pp 12–16. ISSN: 1131-9100

Prandi F, Devigili F, Soave M, Di Dtaso U, De Amicis R (2015) 3D web visualization of huge CityGML models. ISPRS Int Arch Photogramm Remote Sens Spatial Inf Sci 3/W3:601–605. https://doi.org/10.5194/isprsarchives-XL-3-W3-601-2015

Prieto I, Izkara J, Béjar R (2016) Web-Based tool for the sustainable refurbishment in historic districts based on 3D city model. Advances in 3D Geoinformation, Lecture Notes in Geoinformation and Cartography, 159. https://doi.org/10.1007/978-3-319-25691-7_9

Rodrigues, J., Figueiredo, M., Costa, C. (2013). Web3DGIS for city models with cityGML and X3D. Proceedings of the 17th International Conference on Information Visualisation, July 16-18, London, UK. https://doi.org/10.1109/IV.2013.102

Sierra A (2013) GeoJSON y TopoJSON: comparación entre los formatos de intercambio de Información Geográfica alternativos a GML. IV Jornadas Ibéricas de Infraestructura de Datos Espaciales, Toledo

Sun K, Zhu Y, Pan P, Hou Z, Wang D, Li W, Song J (2019) Geospatial data ontology: the semantic foundation of geospatial data integration and sharing. Big Earth Data 3:269–296. https://doi.org/10.1080/20964471.2019.1661662

Tang F, Yu X, Bell S, Yu H, Zeng W, Phung T, Natcher D (2019) A web GIS platform for environmental livelihood value assessment in northeastern British Columbia. CEUR Workshop Proceedings 6: 2323. https://doi.org/10.1007/s10109-004-0127-2

Valencia J, Muñoz-Nieto A (2018) Infraestructuras de Datos Espaciales tridimensionales. Hacia un modelo real de la información geográfica. Mapping 26(186):56–65

Yali LU, Huijie Z (2018) Three-dimensional campus 360-degree video encoding VR technology based on OpenGL. Multimedia Tools and Applications, 1-9. https://doi.org/10.1007/s11042-018-6306-9

Yao Z, Nagel C, Kunde F, Hudra G, Willkomm P, Donaubauer A, Adolphi T, Kolbe TH (2018) 3DCityDB - a 3D geodatabase solution for the management, analysis, and visualization of semantic 3D city models based on CityGML. Open Geospatial Data Softw Standards 3(2). https://doi.org/10.1186/s40965-018-0046-7

Zhu, G., Qiao, W. (2013). Design and realization of thematic mapping in Web 3 DGIS. GEOProcessing 2013: The Fifth International Conference on Advanced Geographic Information Systems, Applications, and Services

Webs

3DCityDB (2020) https://www.3dcitydb.org/3dcitydb/. Accessed 10 Apr 2020

3DGIS (2020) https://www.3dgis.it/it/. Accessed 10 Apr 2020

Amazon Web Service (2020) https://aws.amazon.com/es/. Accessed 10 Apr 2020

Aristoteles, koblenz-landau University (2020) https://www.uni-koblenz-landau.de/de/koblenz/fb4/uebergreifend/er/stormodelling/tools/aristoteles. Accessed 10 Apr 2020

BlueBird (2020) http://www.bluebirdcorp.com. Accessed 10 Apr 2020

BodyParser, npm (2020) https://www.npmjs.com/package/body-parser. Accessed 10 Apr 2020

Carto (2020) https://carto.com/. Accessed 10 Apr 2020

DG Catastro (2020) http://www.catastro.meh.es/. Accessed 10 Apr 2020

Cesium JS (2020) https://cesiumjs.org/. Accessed 10 Apr 2020

Citygml2vrml converter (2020) https://code.google.com/archive/p/libcitygml/wikis/citygml2vrml.wiki. Accessed 10 Apr 2020

Cityvu, 3DGIS (2020) https://www.3dgis.it/it/prodotto/cityvu-visualizzatore-3d-geografico/. Accessed 10 Apr 2020

COLLADA, wiki (2020) https://en.wikipedia.org/wiki/COLLADA. Accessed 10 Apr 2020

CORS, W3C (2020) https://www.w3.org/TR/cors/. Accessed 10 Apr 2020

Express JS (2020) https://expressjs.com/es/. Accessed 10 Apr 2020

FZK Viewer, KIT (2020) https://www.iai.kit.edu/1302.php. Accessed 10 Apr 2020

GeoJSON (2020) https://geojson.org/. Accessed 10 Apr 2020

Geojsplit (2020) https://www.npmjs.com/package/geojsplit. Accessed 10 Apr 2020

GeoServer (2020) http://geoserver.org/. Accessed 10 Apr 2020

GeoTools (2020) https://geotools.org/. Accessed 10 Apr 2020

gTIFF (2020) https://gdal.org/drivers/raster/gtiff.html. Accessed 10 Apr 2020

IEEE standard 29148 (2020) https://standards.ieee.org/standard/29148-2011.html. Accessed 10 Apr 2020

IEEE standard 830 (2020) https://standards.ieee.org/standard/830-1998.html. Accessed 10 Apr 2020

ISO 19115 (2020) https://www.iso.org/standard/53798.html. Accessed 10 Apr 2020

JQuery (2020) https://jquery.com/. Accessed 10 Apr 2020

KML (2020) https://developers.google.com/kml/documentation/kml_tut?hl=es-419. Accessed 10 Apr 2020

Kml2Geojson, Phython (2020) https://pypi.org/project/kml2geojson/. Accessed 10 Apr 2020

LandXplorer, Autodesk (2020) http://download.autodesk.com/us/landxplorer/docs/ldx_citygml_viewer/html/index.html?topic.htm. Accessed 10 Apr 2020

Leaflet JS (2020) https://leafletjs.com/. Accessed 10 Apr 2020

MapBox (2020) https://www.mapbox.com/

MongoDB (2020) https://www.mongodb.com/es. Accessed 10 Apr 2020

Mongoose (2020) https://mongoosejs.com/. Accessed 10 Apr 2020
Node JS (2020) https://nodejs.org/es/. Accessed 10 Apr 2020
OGC JSON (2020) OGC JSON encoding extension to moving features standard. https://www.opengeospatial.org/standards/requests/196. Accessed 10 Apr 2020
Open Geoespatial Consortium (OGC) (2020) http://www.opengeospatial.org/. Accessed 10 Apr 2020
OpenLayers (2020) https://openlayers.org/. Accessed 10 Apr 2020
Oracle Spatial (2009), https://docs.oracle.com/cd/B28359_01/appdev.111/b28400/title.htm. Accessed 10 Apr 2020.
PNOA, IGN (2020) http://www.ign.es/wms-inspire/pnoa-ma? Accessed 10 Apr 2020
PostGree (2020) PostGree/PostGIS versión 2.0 https://postgis.net/2013/08/17/postgis-2-1-0/. Accessed 10 Apr 2020
Python (2020) https://www.python.org/. Accessed 10 April 2020.
Spreadscheets (2020) https://docs.google.com/spreadsheets/u/0/. Accessed 10 Apr 2020
TerraServer (2020) https://www.terraserver.com/. Accessed 10 Apr 2020
Tree JS (2020) https://threejs.org/. Accessed 10 Apr 2020
VRML (2020) https://www.ecured.cu/VRML. Accessed 10 Apr 2020
W3DS (2020) http://w3ds.org/doku.php. Accessed 10 Apr 2020
Web3D Consortium (2020) https://www.Web3d.org/. Accessed 10 Apr 2020
X3D (2020) http://www.Web3d.org/x3d/what-x3d. Accessed 10 Apr 2020